Verification of Directed Acyclic Ad Hoc Networks

Parosh Abdulla, Mohamed Atig, Othmane Rezine

Verification of Directed Acyclic Ad Hoc Networks

Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Othmane Rezine

Uppsala University

Abstract. We study decision problems for parameterized verification of a formal model of ad hoc networks. We consider a model in which the network is composed of a set of processes connected to each other through a directed acyclic graph. Vertices of the graph represent states of individual processes. Adjacent vertices represent single-hop neighbors. The processes are finite-state machines with local and synchronized broadcast transitions. Reception of a broadcast is restricted to the immediate neighbors of the sender process. The underlying connectivity graph constrains communication pattern to only one direction. This allows to model typical communication patterns where data is propagated from a set of central nodes to the rest of the network, or alternatively collected in the other direction. For this model, we consider decidability of the control state reachability (coverability) problem, defined over two classes of architectures, namely the class of all acyclic networks (for which we show undecidability) and that of acyclic networks with a bounded depth (for which we show decidability). The decision problems are parameterized both by the size and by the topology of the underlying network.

1 Introduction

The analysis and verification of models for wireless ad hoc networks have attracted much interest in recent years [4, 11, 2, 3, 9, 8, 12, 13, 10]. Such networks usually consist of arbitrary numbers of nodes that communicate wirelessly in arbitrarily configured networks. Several features in their behaviors make them both attractive and difficult from the point of view of verification. First, the network infrastructure can be static or dynamic but is usually not a priori defined. Also, the communication between nodes occurs via broadcast over the shared radio channel medium. Messages broadcasted by a given node are only received by nodes in its proximity, in contrast to classical broadcast communication in which *all* processes of the system are able to receive the sent messages. Furthermore, since the systems may contain unbounded numbers of processes, and since the protocols are supposed to work independently from specific configurations of the network, we need to perform *parameterized verification* where we prove correctness of the system regardless of the number of nodes or the topology of the network.

Using a model similar to that proposed in [2], we view the network as a graph of nodes, where each node runs an instance of a given finite-state process. The

graph defines the underlying connectivity of the network, while a process models the code of a given fixed protocol that runs on the node. For such networks, the behavior of a node can in general be specified in terms of a sequence of states with transitions corresponding to local or to broadcast operations. Local transitions are internal to a node and do not affect the states of the other nodes in the network. Broadcast transitions on the other hand may have an impact on other nodes in the network. More precisely, we consider *selective broadcast* transitions that involve a sender (the broadcasting node), together with a set of receivers composed of the nodes that are in the topological vicinity of the sender, and that are willing to receive the broadcasted message. The vicinity of a node is defined by the underlying communication graph of the network. The broadcasting and the reception of the message happens synchronously for all involved nodes, i.e., the sender and all potential receivers in its vicinity. The interleaving semantics of our formalism does not take into account problems that could arise at the physical and link layer, such us message collision for example. We are here more interested in network and application layer protocols where these type of problems are abstracted away.

As argued in [2, 3], the *control state reachability problem* or the *coverability problem* seems to be adequate for capturing several interesting properties that arise in parameterized verification of ad hoc networks. The problem consists in checking whether the system can start from a given initial configuration and evolve to reach a configuration in which at least one of the processes is in a given state. Since we are performing parameterized verification, the number of nodes that has to be handled in the analysis is not a priori bounded. In other words, we are dealing with the verification of an infinite-state system. Indeed, it is shown in [2] that the coverability problem is undecidable in the general case. Therefore, an important line of work has been done to identify classes of network topologies for which algorithmic verification is at least theoretically possible [3]. This paper proposes one such a class of topologies where the underlying graph is acyclic, and hence the communication from a node to another goes only through one direction. Such patterns arise, for instance, in the context of *Wireless Sensor Networks* (Wsn), where small wireless devices are distributed over an area in order to perform different types of measurements (temperature, humidity, etc.). In Wsn, it is common that the topology is static over time. Furthermore, it is also common in Wsn that communication follows a specific direction; for instance this is the case for flooding protocols at the network layer [7], and for the optimized *directed diffusion protocol* [6].

From the verification point of view, we show that the coverability problem is undecidable even in the case where the network topology is acyclic (section 5). We show the undecidability result through a reduction from an undecidable problem for finite-state transducers (section 4). Then, we consider a restricted version of the problem where we assume that the depth of the acyclic graph is bounded by a given natural number k. In fact, we are still dealing with an infinite state-system since we may have an unbounded number of nodes, and an unbounded in- and out-degrees for the nodes of the graph. For this case we show

decidability of the coverability problem. The proof is carried out in several steps. First we reduce the problem from the case of general acyclic graphs to that of inverted forests (forests with all edges reversed) and then to the case of inverted trees (section 6). For the case of inverted trees, we propose a novel symbolic representation of infinite sets of configurations. This symbolic representation amounts to having "higher-order multisets" in which a multiset of a certain order contains multisets of lower orders (section 7). We show that this allows to define a symbolic backward reachability analysis based on a non-trivial instantiation of the the framework of well quasi-ordered transition systems [1].

2 Preliminaries

In this section, we introduce some basic definitions and notations that we will use in the rest of the paper.

We use $\mathbb{N}$ and $\mathbb{N}_{>0}$ to denote the sets of natural numbers and positive natural numbers, respectively. Given a finite set A, we use $|A|$ to denote the number of elements in A. We use $A^{\otimes}$ to denote the set of finite multisets over A; and use A^* to denote the set of finite words over A. For words $w_1, w_2 \in A^*$ we use $w_1 \cdot w_2$ to denote the concatenation of w_1 and w_2. Sometimes, we write multisets as lists, e.g., $[a, a, b, b, b]$ is a multiset with two occurrences of a and three occurrences of b. A *quasi-ordering* (*ordering* for short) $\sqsubseteq$ on a set A is a reflexive and transitive binary relation over A (i.e. $\sqsubseteq \subseteq A \times A$, $a \sqsubseteq a$ and $a \sqsubseteq b, b \sqsubseteq c \Rightarrow a \sqsubseteq c$ for any $a, b, c \in A$). We extend the ordering $\sqsubseteq$ on A to an ordering $\sqsubseteq^{\otimes}$ on the set $A^{\otimes}$ of multisets over A such that $[a_1, \ldots, a_m] \sqsubseteq^{\otimes} [b_1, \ldots, b_n]$ if there is an injection $h : \{1, \ldots, m\} \mapsto \{1, \ldots, n\}$ with $a_i \sqsubseteq b_{h(i)}$ for all $i : 1 \leqslant i \leqslant m$. Given a function $f : A \mapsto \mathbb{N}$, we define $\max(f) := max\{f(e) | e \in A\}$ to be the largest value taken by f over A. For a function $f : A \mapsto B$, we use $f\,[a \leftarrow b]$ to denote the function f' such that $f'(a) = b$ and $f'(a') = f(a')$ if $a' \neq a$.

A *(directed) graph* is a pair $G = \langle V, E \rangle$ where V is a finite set of *vertices* and $E \subseteq V \times V$ is the set of *edges*. Two graphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ are said to be disjoint iff $V_1 \cap V_2 = \varnothing$. For vertices $u, v \in V$, we use $u \rightsquigarrow_G v$ to denote that $\langle u, v \rangle \in E$, use $\overset{*}{\rightsquigarrow}_G$ to denote the reflexive transitive closure of $\rightsquigarrow_G$, and use $\overset{+}{\rightsquigarrow}_G$ to denote the transitive closure of $\rightsquigarrow_G$. A *path* in G is a finite sequence $\pi = v_1 v_2 \cdots v_k$ where $v_i \rightsquigarrow_G v_{i+1}$ for all $i : 1 \leqslant i < k$. We define $first(\pi) := v_1$ and $last(\pi) := v_k$. Notice that $u \overset{*}{\rightsquigarrow}_G v$ iff there is a finite path in G with $first(\pi) = u$ and $last(\pi) = v$. For a vertex $v \in V$, we define $succ_G(v) := \{u | \, v \rightsquigarrow_G u\}$ to be its set of *successor* vertices, and define $pred_G(v) := \{u | \, u \rightsquigarrow_G v\}$ to be its set of *predecessor* vertices. For a graph $G = \langle V, E \rangle$, we define its transpose $G^{Transp} := \langle V, E^{Transp} \rangle$, where $E^{Transp} := \{\langle v, u \rangle | \, \langle u, v \rangle \in E\}$. In other words G^{Transp} is G with all edges reversed. We say that G is a DAG if there are no cycles in G, i.e., there are no vertices $v \in V$ with $v \overset{+}{\rightsquigarrow}_G v$. Fix a DAG $G = \langle V, E \rangle$. We define $\#G := |\{v \in V | \, |succ_G(v)| > 1\}|$. In other words, it is the number of vertices in the graph whose set of successors contains more than one element. For a vertex $v \in V$, we define $height_G(v) := 0$ if $succ_G(v) = \varnothing$, and define $height_G(v) := 1 + \max_{u \in succ_G(v)}(height_G(u))$

otherwise. We define $\mathit{height}(G) := \max_{v \in V} \mathit{height}_G(v)$, i.e., it is the length of a longest path in G. We define $\mathit{depth}_G(v) := 0$ if $\mathit{pred}_G(v) = \varnothing$, and define $\mathit{depth}_G(v) := 1 + \max_{u \in \mathit{pred}_G(v)}(\mathit{depth}_G(u))$ otherwise. We define $\mathit{depth}(G) := \max_{v \in V} \mathit{depth}_G(v)$. A *leaf* of G is a vertex $v \in V$ with height zero, i.e., $\mathit{succ}_G(v) = \varnothing$. We use $\mathit{leaves}(G)$ to denote the set of leaves of G. A *forest* is a DAG such that for all distinct pairs of vertices $v, u \in V$ we have $\mathit{succ}_G(v) \cap \mathit{succ}_G(u) = \varnothing$. A *tree* is a forest such that $|\{v|\ \mathit{pred}_G(v) = \varnothing\}| = 1$. We say that G is an *inverted forest/tree* if G^{Transp} is a forest/tree. The *root* of an inverted tree G is the unique vertex v with $\mathit{succ}_G(v) = \varnothing$. Notice that a DAG G is an inverted forest iff $\#G = 0$, i.e., it does not contain any vertices with multiple successors.

3 Directed Acyclic Ad-Hoc Networks

A Directed Acyclic Ad-Hoc Network (DAAHN) contains a finite (but arbitrary) number of nodes that are organized in a DAG. The vertices of the DAG represent individual processes, while the DAG models the topology of the network. The processes are modeled as finite-state automata that can perform both local and synchronized broadcast transitions. The successors of a vertex are the set of processes that are able to "hear" broadcast messages issued by the vertex. Depending on its local state, a successor may participate in the broadcast transition or not. Below, we describe the syntax and the operational semantics of a DAAHN, and then define two decision problems for the model related to reachability properties.

Syntax. An *Ad-Hoc Network* (AHN) consists of a pair $N = \langle P, G \rangle$ where P is a finite-state automaton describing the behavior of each process, and $G = \langle V, E \rangle$ is the communication graph between the processes. A *process* P is a tuple $\langle Q, \Sigma, \Delta, q_{init} \rangle$ where Q is a finite set of states, Σ is a finite message alphabet, $q_{init} \in Q$ is the initial state, and $\Delta \subseteq Q \times (\{\tau\} \cup \{b(m)|\ m \in \Sigma\} \cup \{r(m)|\ m \in \Sigma\}) \times Q$ is the transition relation. Intuitively, τ represents a local (internal) transition of the process. The operation $b(m)$ corresponds to broadcasting a message m, while $r(m)$ corresponds to receiving the message m. We say that N is a DAAHN if G is a DAG.

Operational Semantics. We give the operational semantics by defining the transition system induced by N. A *configuration* c of N is a function $c : V \mapsto Q$ that defines, for each vertex $v \in V$ (i.e., a process position), a state $q \in Q$. We use $q \in c$ to denote that there exists a vertex $v \in V$ such that $c(v) = q$. We use C to denote the set of configurations of N, and define the *initial configuration* c_{init} such that $c_{init}(v) = q_{init}$ for all $v \in V$. We define a transition relation $\rightarrow_N$ on the set C by $\rightarrow_N := \bigcup_{t \in \Delta} \xrightarrow{t}$, where $\xrightarrow{t}$ describes the effect of performing the transition t. Given two configurations $c, c' \in C$, we have $c \xrightarrow{t}_N c'$ if one of the following conditions holds:

- *Local transition.* There is a $v \in V$ such that $t = \langle c(v), \tau, c'(v)\rangle \in \Delta$ and for every $v' \in V \backslash \{v\}$, we have that $c'(v') = c(v')$. A local transition modifies only the state of the involved process.
- *Broadcast.* There are $v \in V$ and $m \in \Sigma$ such that $t = \langle c(v), b(m), c'(v)\rangle \in \Delta$, and for every $v' \in V \backslash \{v\}$ one of the following conditions holds:
  - $v \rightsquigarrow_G v'$ and $\langle c(v'), r(m), c'(v')\rangle \in \Delta$.
  - $v \rightsquigarrow_G v'$, $\langle c(v'), r(m), q\rangle \notin \Delta$ for any $q \in Q$, and $c'(v') = c(v')$.
  - $v \not\rightsquigarrow_G v'$ and $c'(v') = c(v')$.

  In a broadcast transition, any successor of the sender process that can receive the message m is obliged to participate in the transition.

For both types of transitions, the topology of the system is not affected. We use $\xrightarrow{*}_N$ to denote the reflexive transitive closure of $\rightarrow_N$. A (finite) *run* ρ of N is a sequence $c_0 c_1 \ldots c_n$ of configurations such that $c_0 = c_{init}$ and $c_i \rightarrow_N c_{i+1}$ for $i : 0 \leqslant i < n$. We use $last(\rho)$ to denote c_n. A configuration c is said to be *reachable* in N if there is a run ρ of N such that $last(\rho) = c$ (notice that this is equivalent to $c_{init} \xrightarrow{*}_N c$). A state $q \in Q$ is said to be reachable in N if $q \in c$ for some reachable configuration c.

Decision Problems. The *state reachability problem* or *coverability problem* Cover is defined by a process $P = \langle Q, \Sigma, \Delta, q_{init}\rangle$ and a state *target* $\in Q$. The task is to check whether there is a Dag G such that *target* is reachable in the Daahn $N = \langle P, G\rangle$. The *bounded state reachability problem* Bounded-Cover is defined by a process $P = \langle Q, \Sigma, \Delta, q_{init}\rangle$, a state *target* $\in Q$, and a natural number $k \in \mathbb{N}$. The task is to check whether there is a Dag G with $height(G) \leqslant k$ such that *target* is reachable in the Daahn $N = \langle P, G\rangle$.

4 Transducers

We recall the standard definition of transducers and an undecidable problem for them. A *(finite-state) automaton* is a tuple $A = \langle Q, \Sigma, \Delta, q_{init}, Q_{final}\rangle$ where Q is a finite set of states, Σ is a finite alphabet, $q_{init} \in Q$ is the initial state, $Q_{final} \subseteq Q$ is the set of final states, and $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation. We define the language $L(A)$ of A as usual. A *(finite-state) transducer* $T = \langle Q, \Sigma, \Delta, q_{init}, Q_{final}\rangle$ is of the same form as a finite-state automaton except that $\Delta \subseteq Q \times \Sigma \times \Sigma \times Q$. Thus, a member of $L(T)$ is a word of pairs over Σ, i.e., a member of $\left(\Sigma^2\right)^*$. A transducer T induces a binary relation $R(T)$ on the set Σ^* such that $\langle a_1 \cdots a_n, b_1 \cdots b_n\rangle \in R(T)$ if $\langle a_1, b_1\rangle \cdots \langle a_n, b_n\rangle \in L(T)$. For a word $w \in \Sigma^*$, we define $T(w) := \{v \mid \langle w, v\rangle \in R(T)\}$. For a set W of words, we define $T(W) := \cup_{w \in W} T(w)$. Given an automaton A and a transducer T (with identical alphabets Σ) we define $T(A) := T(L(A))$. For a natural number $i \in \mathbb{N}$ and a word $w \in \Sigma^*$, we define $T^i(w)$ inductively by $T^0(w) := \{w\}$ and $T^{i+1}(w) := T(T^i(w))$. In other words, it is the result of i applications of the relation induced by T on w. We extend the definition of T^i to sets of words in the expected manner. For an automaton A, we define $T^i(A) := T^i(L(A))$.

An instance of the problem TRANSD consists of two automata A and B, and a transducer T, all with identical alphabets Σ. The task is to check whether there is an $i \in \mathbb{N}$ such that $T^i(A) \cap L(B) \neq \emptyset$. It is straightforward to show undecidability of TRANSD through a reduction from a certain non-trivial problem for Turing machines, namely whether a given Turing machine M will eventually print a given symbol a on its tape. More precisely, we use $L(A)$ to describe an appropriate encoding of (i) an empty tape of M, and (ii) the initial position of its head on the tape. The transducer T encodes one move of M, by non-deterministically guessing the position of the head, and then (i) moving the head, (ii) changing one symbol on the tape, and (iii) changing its state, according to the transition relation of M. Finally, the automaton B accepts all words that contains the symbol a.

5 Undecidability of Cover

In this section, we prove the following theorem.

Theorem 1. COVER *is undecidable.*

We show undecidability through a reduction from TRANSD to COVER. Consider an instance of TRANSD defined by automata $A = \left\langle Q^A, \Sigma_A, \Delta^A, q^A_{init}, Q^A_{final} \right\rangle$ and $B = \left\langle Q^B, \Sigma_B, \Delta^B, q^B_{init}, Q^B_{final} \right\rangle$, and transducer $T = \left\langle Q^T, \Sigma_T, \Delta^T, q^T_{init}, Q^T_{final} \right\rangle$ (with $\Sigma_A = \Sigma_B = \Sigma_T$). We define a process $P = \langle Q, \Sigma, \Delta, q_{init} \rangle$ and a state $q_{accept} \in Q$ such that there is a DAG G with q_{accept} reachable in the DAAHN $N = \langle P, G \rangle$ iff there is an $i \in \mathbb{N}$ such that $T^i(A) \cap L(B) \neq \emptyset$. The manner in which we define process P (see below) will allow it to simulate both automata A and B and transducer T. The set Q of states of P is defined to be the union of four disjoint sets $Q := \{q_{init}, q_{error}, q_{accept}\} \cup S_A \cup S_B \cup S_T$ described below. The idea of the simulation is that a group of processes in N tries to build a "chain" (of some size, say i), where the root of the chain simulates A, the $(i-2)$ processes in the middle of the chain simulate T, and the last process simulates B. We will refer to such a chain as *transduction chain* below.

Simulating A. The states in S_A are used by P to simulate the automaton A. Each state $q \in Q^A$ in A has a copy $[q]_A$ in S_A. At state q_{init}, the process P may decide to simulate the automaton A (Figure 1), thus becoming the first vertex in a potential transduction chain. It does this by performing the transition $\left\langle q_{init}, b(A_{start}), \left[q^A_{init}\right]_A \right\rangle \in \Delta$ in which it moves to (the copy of) of the initial state of A. At the same time, it issues a broadcast message $b(A_{start})$ to notify its successor processes in G that it has started the simulation of A. For each transition $\langle q_1, a, q_2 \rangle \in \Delta^A$ in A there is a transition $\langle [q_1]_A, b(a), [q_2]_A \rangle \in \Delta$ in which P simulates changing of states in A and broadcasts the symbol a to its successors. Finally, for each final state $q \in Q^A_{final}$, there is a transition

$\left\langle [q]_A, b(m_{end}), q^A_{end} \right\rangle \in \Delta$ in which P declares that it has ended the simulation of A (by broadcasting the message m_{end}), after which P stops (there are no outgoing transition from q^A_{end}). Thus, in this mode, the process P broadcasts a sequence of messages corresponding to a word in $L(A)$ followed by the end-marker m_{end}.

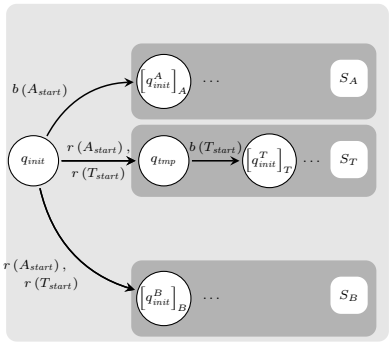


Fig. 1. Process P: initial choices

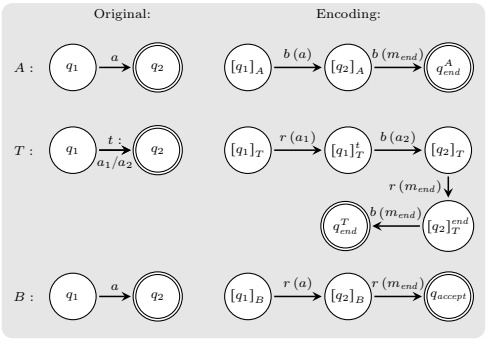


Fig. 2. Transition and accepting state encoding

Simulating T. The states in S_T are used by P to simulate the transducer T. Each state $q \in Q^T$ in T has several corresponding states in S_T. More precisely, it has one copy $[q]_T$ (as in the case of A above); together with one temporary state $[q]^t_T$ for each transition $t = \langle q, a_1, a_2, q' \rangle \in \Delta^T$, i.e., for each transition whose source state is q. At state q_{init}, if the process P receives a message A_{start} or T_{start} from one of its predecessors, then it may decide to simulate the transducer T (Figure 1). It does so by (i) first performing one of the transitions $\langle q_{init}, r(A_{start}), q_{tmp} \rangle \in \Delta$ and $\langle q_{init}, r(T_{start}), q_{tmp} \rangle \in \Delta$, where $q_{tmp} \in S_T$ is a temporary state, followed by (ii) performing the transition $\left\langle q_{tmp}, b(T_{start}), \left[q^T_{init}\right]_T \right\rangle \in \Delta$ in which it moves to the first copy of the initial state of T. At the same time, it issues a broadcast message $b(T_{start})$ to its successors declaring that it has started the simulation of T. Intuitively, if P has received A_{start}, it will be the second process in a transduction chain (its predecessor will be the first since it simulates A), while if it has received T_{start}, it will be the $(k+1)$-th process in the chain (its predecessor will be the k-th process and the predecessor also simulates T). For each transition $t = \langle q_1, a_1, a_2, q_2 \rangle \in \Delta^T$ in T there are two transitions $\left\langle [q_1]_T, r(a_1), [q_1]^t_T \right\rangle \in \Delta$ and $\left\langle [q_1]^t_T, b(a_2), [q_2]_T \right\rangle \in \Delta$. Here, P receives the message a_1 from its predecessor (in the chain), and sends a_2 to its successors. Although, a node may have several predecessors, only one of them is allowed to act as the predecessor of the current node in the chain. This is ensured by transitions $\langle q, r(A_{start}), q_{error} \rangle \in \Delta$ and $\langle q, r(T_{start}), q_{error} \rangle \in \Delta$ for each $q \in S_T$. In other words, if the current process has already received a message from one predecessor (and thus moved to a state in S_T) then it moves to the state q_{error} if it later receives messages from any of its other predecessors. The process P then immediately suspends the simulation (there are no outgoing transition from q_{error}). Also, the process is not

allowed to be "disturbed" by its predecessor while it is in the temporary state q_{tmp} or in one of the temporary states of the form $[q]^t_T$. This is due to the fact that the process in such a temporary state has not yet had time to perform the next broadcast, and therefore it is not yet ready to receive the next message form the predecessor (if this is not done, such a message will be lost in the simulation). To encode this, we add extra transitions $\langle q, r(a), q_{error}\rangle$ for each temporary state q and each message a. Also, in order to discard any sequence of received messages that do not correspond to a valid T input word, we add in Δ the transition $\langle [q]_T, r(a), q_{error}\rangle$ for every state q of Q^T and for every message $a \in \Sigma_T$ such that there is no $q' \in Q^T$ such that $\langle q, a, q'\rangle \in \Delta^T$. Finally, for each final state $q \in Q^T_{final}$, there is a transition $\left\langle [q]_T, r(m_{end}), [q]^{end}_T \right\rangle \in \Delta$; and a transition $\left\langle [q]^{end}_T, b(m_{end}), q^T_{end} \right\rangle \in \Delta$ (where $[q]^{end}_T$ is a temporary state). If the process happens to be in a final state, and it receives the end-marker from its predecessor in the chain, then it ends its simulation by notifying its successor and moving to the state q^T_{end}. Thus, in this mode, the process P receives a word w from its predecessor and sends a word in $T(w)$ to its successor.

Simulating B. The states in S_B are used by P to simulate the automaton B. Each state $q \in Q^B$ in B has a copy $[q]_B$ in S_B. At state q_{init}, if the process P receives a message A_{start} or T_{start} from one of its predecessors, then it may decide to simulate the automaton B (Figure 1). It does so by performing one of the transitions $\left\langle q_{init}, r(A_{start}), \left[q^B_{init}\right]_B \right\rangle \in \Delta$ and $\left\langle q_{init}, r(T_{start}), \left[q^B_{init}\right]_B \right\rangle \in \Delta$. In either case, it moves to the (copy of) the initial state of B. For each transition $\langle q_1, a, q_2\rangle \in \Delta^B$ in B there is a transition $\langle [q_1]_B, r(a), [q_2]_B\rangle$ in which P simulates the changing of states in B and receives the symbol a from its predecessor. In a similar manner to the case of T, we also add transitions $\langle q, r(A_{start}), q_{error}\rangle \in \Delta$ and $\langle q, r(T_{start}), q_{error}\rangle \in \Delta$ for each $q \in S_B$, and $\langle [q]_B, r(a), q_{error}\rangle \in \Delta$ for every state q of B and for every message $a \in \Sigma_B$ such that there is no $q' \in Q^B$ such that $\langle q, a, q'\rangle \in \Delta^B$. Finally, for each final state $q \in Q^B_{final}$, there is a transition $\langle [q]_B, r(m_{end}), q_{accept}\rangle \in \Delta$ in which P ends the simulation of B. Thus, in this mode, the process P receives a sequence of messages corresponding to a word in $L(B)$ followed by the end-marker m_{end}. In such a case, the process moves to the state q_{accept} which means that the given instance of Cover has a positive solution.

Correctness. We show correctness of our reduction. Suppose that the given instance of Transd has a positive answer, i.e., there is an $i \in \mathbb{N}$ and a word $w \in L(A)$ such that $T^i(w) \in L(B)$. We show that there is a Dag G such that q_{accept} is reachable in the Daahn $N = \langle G, P\rangle$, where P is defined as described above. We define $G := \langle \{v_1, v_2, \ldots, v_{i+2}\}, E\rangle$, where $v_j \rightsquigarrow_G v_k$ iff $1 \leqslant j \leqslant i+1$ and $k = j+1$. In other words, the graph forms a chain with $i+2$ nodes. The process at node 1 starts simulating A eventually broadcasting the word w followed by m_{end}. The process at node 2 starts simulating T receiving the word w symbol by symbol, and eventually broadcasting the word $T(w)$ followed by m_{end}. In general, the process at node j for $j : 2 \leqslant j \leqslant i+1$ starts simulating T

receiving the word $T^{j-2}(w)$ symbol by symbol, and eventually broadcasting the word $T^{j-1}(w)$ followed by m_{end}. Finally, the process at node $i+2$ starts simulating B receiving the word $T^i(w)$ symbol by symbol, and eventually moving to the state q_{accept}.

Suppose that the given instance of Cover has a positive answer, i.e., there is a Dag G such that q_{accept} is reachable in the Daahn $N = \langle G, P \rangle$. We show that there is an $i \in \mathbb{N}$ and a word $w \in L(A)$ such that $T^i(w) \in L(B)$. We do this by extracting a transduction chain. We extract the chain vertex by vertex starting by identifying the process that simulates B, then identifying the ones that simulate T, and finally identifying the one that simulates A. Recall that q_{accept} can only be reached in a process that is simulating B. Recall also that such a process can reach q_{accept} if it receives the end-marker from a predecessor process. On the other hand, it cannot receive start messages from two different predecessors before it reaches q_{accept} since this would mean that it would move to the error state q_{error} from which it cannot reach q_{accept}. This implies that the current process has a unique predecessor. Recall that the predecessor, a sending process, must be either a process simulating A or T. If the predecessor is simulating A then we can close the chain, otherwise we have found the next transducer. In the latter case, we repeat the reasoning and find the predecessor again. Let j be the length of the chain obtained in this manner ($j \geqslant 2$ since it contains at least two vertices simulating A resp. B). Define i in the instance of Transd to be $j-2$.

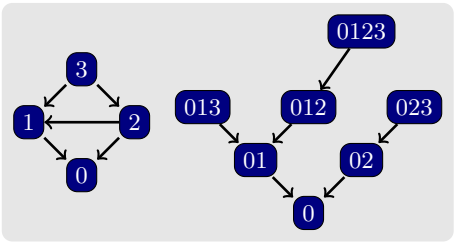
6 Forest Bounded Coverability

In this section, we show that the bounded coverability problem can be reduced from the general case of Dags to the case where we assume the Dag to be an inverted tree. We do that in two steps, namely by first reducing the problem to the case of inverted forests and then to trees.

Forests. A Daahn N is said to be an *inverted forest* if the underlying graph is an inverted forest. We consider a restricted version of Bounded-Cover, which we call Forest-Bounded-Cover. In Forest-Bounded-Cover, we require that the given Daahn is an inverted forest. We show the following theorem.

Theorem 2. Bounded-Cover *is reducible to* Forest-Bounded-Cover.

In order to prove this theorem, we first introduce a split operator over Dags. Consider a Dag $G = \langle V, E \rangle$. The *split* operator splits the nodes of G, transforming it into an inverted forest. We define an inverted forest $G^\bullet := \langle V^\bullet, E^\bullet \rangle$ as follows. Each vertex $v \in V$ induces a set $v^\bullet$ in $V^\bullet$. A member of $v^\bullet$ is an inverted path π in G with $first(\pi) \in leaves(G)$ and $last(\pi) = v$.



The set $v^\bullet$ is defined using induction on the height of v as follows. If $height_G(v) = 0$ (i.e., v is a leaf) then $v^\bullet := \{v\}$. Otherwise, $v^\bullet := \{\pi \cdot v |\ \exists u.\, v \rightsquigarrow_G u \wedge \pi \in u^\bullet\}$. In other words, we split v into a number of copies, each corresponding to a path starting from a successor of v and ending in a leaf in G. We define $E^\bullet := \{\langle \pi_1, \pi_2 \rangle |\ \pi_1 = \pi_2 \cdot v\}$. Notice that $height_{G^\bullet}(u) = height_G(v)$ for every $v \in V$ and $u \in v^\bullet$. Therefore, $height(G^\bullet) = height(G)$. Furthermore, by definition, any vertex in $G^\bullet$ has at most one successor (no successors if it is of the form $v \in V$, or the unique successor π if is of the form $\pi \cdot v$). This means that $G^\bullet$ is an inverted forest.

Consider an instance of Bounded-Cover defined by $P = \langle Q, \Sigma, \Delta, q_{init} \rangle$, a state *target* $\in Q$, and a natural number $k \in \mathbb{N}$. We claim that the instance of Forest-Bounded-Cover defined by $P = \langle Q, \Sigma, \Delta, q_{init} \rangle$, *target*, and k is equivalent. For a configuration c in $N = \langle P, G \rangle$, we define $c^\bullet$ to be the configuration of $N^\bullet = \langle P, G^\bullet \rangle$ such that $c^\bullet(\pi \cdot v) = c(v)$. The following lemma shows that reachability is preserved by splitting.

Lemma 3. *If* $c_1 \rightarrow_N c_2$ *then* $c_1^\bullet \xrightarrow{*}_{N^\bullet} c_2^\bullet$.

From Lemma 3 and the fact that ${c_{init}}^\bullet(\pi \cdot v) = c_{init}(v) = q_{init}$, we conclude the following:

Lemma 4. *If* c *is reachable in* N *then* $c^\bullet$ *is reachable in* $N^\bullet$.

Now, we are ready to prove Theorem 2. If the given instance of Forest-Bounded-Cover has a positive answer, then the instance of Bounded-Cover has trivially a positive answer (each inverted forest is a Daahn). For the opposite direction, suppose that the instance of Bounded-Cover has a positive answer, i.e., there is a Dag G with $height(G) \leqslant k$ such that *target* is reachable in the Daahn $N = \langle P, G \rangle$. By Lemma 4, we know that *target* is reachable in $\langle P, G^\bullet \rangle$. The result then follows since $G^\bullet$ is an inverted forest and since $height(G^\bullet) = height(G) \leqslant k$.

Trees. We consider a yet more restricted version of Bounded-Cover, which we call Tree-Bounded-Cover. In Tree-Bounded-Cover, we require that the given Daahn is an inverted tree.

Theorem 5. Forest-Bounded-Cover *is reducible to* Tree-Bounded-Cover.

The proof of Theorem 5 is straightforward. Since the nodes inside the tree of a forest do not affect transitions of the nodes inside the other trees, we can solve Tree-Bounded-Cover for each tree separately. The given instance of Forest-Bounded-Cover has a positive answer iff Tree-Bounded-Cover has a positive answer for any of the component trees.

7 Tree Bounded Coverability

In this section, we prove the following theorem.

Theorem 6. TREE-BOUNDED-COVER *is decidable.*

We devote the section to the proof of Theorem 6. To do that, we instantiate the framework of *well quasi-ordered transition systems* introduced in [1]. The main ingredient of this framework is to show that the transition relation induced by the system is *monotonic* wrt. a *well quasi-ordering (wqo)* on the set of configurations. We define an ordering that we denote by $\sqsubseteq$ on configurations that are inverted trees and show monotonicity of the system behavior wrt. this ordering. Unfortunately, it is not possible to apply existing frameworks (such as the one in [1]) to directly prove the wqo of $\sqsubseteq$ on inverted trees. Therefore, we introduce a new ordering that we denote by $\sqsubseteq_2$ on a set of "higher-order multisets". We show that the ordering on higher-order multisets $\sqsubseteq_2$ is indeed a wqo and that it is equivalent to the original ordering $\sqsubseteq$ on inverted trees, which proves that $\sqsubseteq$ is itself a wqo. Then, we recall the basic concepts of the framework of well quasi-ordered systems, and show how the framework can be instantiated to prove Theorem 6.

7.1 Ordering

Assume a process $P = \langle Q, \Sigma, \Delta, q_{init} \rangle$. An *extended configuration* is a pair $e = \langle G, c \rangle$ where G is an inverted tree, and c is a configuration of the DAAHN $\langle G, P \rangle$. We use $\mathbb{E}$ to denote the set of extended configurations, and, for $k \geqslant 1$, we use $\mathbb{E}^k$ to denote the set of extended configurations $\langle G, c \rangle$ where the inverted tree G is of height at most k. We define an ordering on $\mathbb{E}$ as follows. Consider extended configurations $e = \langle G, c \rangle$ with $G = \langle V, E \rangle$ and $e' = \langle G', c' \rangle$ with $G' = \langle V', E' \rangle$. For an injection $\alpha : V \mapsto V'$, we use $e \sqsubseteq^\alpha e'$ to denote that the following two conditions hold for all $v \in V$: (i) $c(v) = c'(\alpha(v))$, and (ii) If $u \rightsquigarrow_G v$ then $\alpha(u) \rightsquigarrow_{G'} \alpha(v)$. We write $e_1 \sqsubseteq e_2$ if $e \sqsubseteq^\alpha e'$ for some α. Intuitively, we can view an extended configuration as an inverted tree that is unranked (a node may have any number of predecessors) and unordered (the order in which the predecessors occur is not relevant). The ordering $e_1 \sqsubseteq e_2$ then means that the inverted tree corresponding to e_1 has a copy (an image) inside the inverted tree corresponding to e_2. In Figure 3, three extended configurations are depicted as inverted trees e_1, e_2, e_3. Here $e_1 \sqsubseteq e_2 \sqsubseteq e_3$.

7.2 Monotonicity

Given a process P, we define a transition relation $\longrightarrow$ on $\mathbb{E}$ where $\langle G, c \rangle \longrightarrow \langle G', c' \rangle$ if $G' = G$, $N = \langle P, G \rangle$, and $c \rightarrow_N c'$. The following lemma shows monotonicity of $\longrightarrow$ wrt. $\sqsubseteq$. Assume that e_1, e_2, e_3 are extended configurations.

Lemma 7. *If* $e_1 \longrightarrow e_2$ *and* $e_1 \sqsubseteq e_3$ *then there is an* e_4 *such that* $e_3 \longrightarrow e_4$ *and* $e_2 \sqsubseteq e_4$.

7.3 Higher-Order Multisets

For a finite set A and $k \geqslant 0$, we define the set $A^{\otimes k}$ inductively as follows: (i) $A^{\otimes 0} := A$; and (ii) $A^{\otimes k+1} := A^{\otimes k} \cup \left(A \times \left(A^{\otimes k}\right)^{\otimes}\right)$. In other words, a higher-order multiset of order 0 is an element in A, while a multiset of order $k+1$ is either a multiset of order k, or a pair consisting of an element in A together with a multiset of multisets of order k. Intuitively, a higher-order multiset defines an inverted tree (corresponding to an extended configuration). More precisely, a higher-order multiset of the form a represents an inverted tree consisting of a single node (labeled by a), while the higher-order multiset $\langle a, [B_1, \ldots, B_k]\rangle$ represents an inverted tree with a root labeled a, and where predecessors of the root are themselves the roots of the inverted subtrees represented by $B_1, \ldots, B_k$ respectively (see Figure 3). We define an ordering $\sqsubseteq_2$ on $A^{\otimes k}$ in two steps. First, we define an ordering $\sqsubseteq_1$ on $A^{\otimes k}$ such that

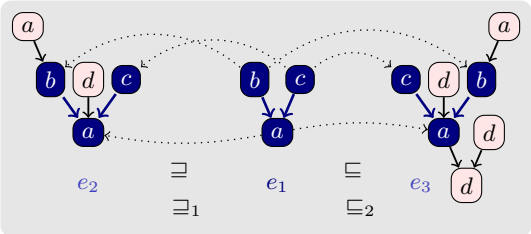


Fig. 3. The extended configurations e_1, e_2, and e_3 correspond to the higher order multisets $B_1 = \langle a, [b, c]\rangle$, $B_2 = \langle a, [\langle b, [a]\rangle, d, c]\rangle$, and $B_3 = \langle d, [\langle a, [\langle b, [a]\rangle, d, c]\rangle, d]\rangle$ respectively.

- $a \sqsubseteq_1 a'$ if $a = a'$; and $a \sqsubseteq_1 \langle a', B\rangle$ if $a = a'$.
- $\langle a, [B_1, \ldots, B_k]\rangle \sqsubseteq_1 \langle a', [B'_1, \ldots, B'_\ell]\rangle$ if $a = a'$ and there is an injection $h : \{1, \ldots, k\} \mapsto \{1, \ldots, \ell\}$ with $B_i \sqsubseteq_1 B'_{h(i)}$ for all $i : 1 \leqslant i \leqslant k$. Notice that the second condition is equivalent to $[B_1, \ldots, B_k] \sqsubseteq_1^{\otimes} [B'_1, \ldots, B'_\ell]$.

Intuitively, $B_1 \sqsubseteq_1 B_2$ means that a copy of the inverted tree corresponding to B_1 occurs in the inverted tree corresponding to B_2 *starting* from the root. For instance, consider B_1, B_2, B_3 in Figure 3. According to the definition of $\sqsubseteq_1$, we have $B_1 \sqsubseteq_1 B_2$ while $B_1 \not\sqsubseteq_1 B_3$. This is reflected in the inverted trees corresponding to the extended configurations e_1, e_2, e_3. Although copies of e_1 occurs both in e_2 and e_3, the copy of e_1 does not start from the root of e_3. Now, we define $\sqsubseteq_2$ as follows.

- $a \sqsubseteq_2 a'$ if $a = a'$; and $a \sqsubseteq_2 \langle a', B\rangle$ if $a = a'$ or $a \sqsubseteq_2 B$.
- $\langle a, [B_1, \ldots, B_k]\rangle \sqsubseteq_2 \langle a', [B'_1, \ldots, B'_\ell]\rangle$ if one of the following two cases is satisfied:
  - $a = a'$ and there is an injection $h : \{1, \ldots, k\} \mapsto \{1, \ldots, \ell\}$ with $B_i \sqsubseteq_1 B'_{h(i)}$ for all $i : 1 \leqslant i \leqslant k$.
  - $\langle a, [B_1, \ldots, B_k]\rangle \sqsubseteq_2 B'_i$ for some $i : 1 \leqslant i \leqslant \ell$.

Notice that $\sqsubseteq_1 \subseteq \sqsubseteq_2$. Intuitively, $B_1 \sqsubseteq_2 B_2$ means that a copy of the inverted tree corresponding to e_1 occur somewhere in the inverted tree corresponding to B_2 (not necessarily starting from the root). In Figure 3, $B_1 \sqsubseteq_2 B_3$ (and a copy of e_1 occurs in e_3).

7.4 Encoding

We define an encoding function # that translates each extended configuration to a higher-order multiset. Formally, consider an extended configuration $\langle G, c\rangle$ with $G = \langle V, E\rangle$. First, we define $\#(v, c)$, for $v \in V$, by induction on $depth_G(v)$ as follows:

- If $depth_G(v) = 0$ then $\#(v, c) := c(v)$. In this case, the encoding is of order 0 (given by the state of the vertex).
- If $depth_G(v) > 0$ then let $pred_G(v) = \{v_1, \ldots, v_n\}$. Then, $\#(v, c) := \langle c(v), [\#(v_1, c), \ldots, \#(v_n, c)]\rangle$. The encoding is of the same order as the depth of the vertex; it consists of the state of the vertex itself together with the multiset of the encodings of its predecessors.

We define $\#e := \#(v, c)$ where v is the root of G. Notice that the order of $\#e$ is identical to the height of the inverted tree G. As an example, in Figure 3, if we view an inverted tree e_i, $i = 1, 2, 3$, as an extended configuration then its encoding is given by B_i. The following lemma shows that the orderings on extended configurations and higher-order multisets coincide. Let e_1 and e_2 be two extended configurations.

Lemma 8. *$e_1 \sqsubseteq e_2$ iff $\#e_1 \sqsubseteq_2 \#e_2$.*

7.5 Well Quasi-Orderings

Let A be a set and let $\sqsubseteq$ be a quasi-ordering on A. We say that $\sqsubseteq$ is *well quasi-ordering (wqo)* if it satisfies the following property: for any infinite sequence $a_0, a_1, a_2, \ldots$ of elements in A, there are $i < j$ with $a_i \sqsubseteq a_j$. We will use the following variant of Higman's Lemma [5] for our purposes:

Lemma 9. *If $\sqsubseteq$ is wqo on A then $\sqsubseteq^{\otimes}$ is a wqo on $A^{\otimes}$.*

Now, we show that the ordering $\sqsubseteq_2$ is a wqo on $A^{\otimes k}$ for any given $k \geqslant 0$. To show $\sqsubseteq_2$ is a wqo, we first show that $\sqsubseteq_1$ is a wqo on $A^{\otimes k}$ for any given $k \geqslant 0$. We use induction on k. The base case is trivial since it amounts to equality being a wqo on a finite alphabet. Consider an infinite sequence $\langle a_0, D_0\rangle, \langle a_1, D_1\rangle, \langle a_2, D_2\rangle, \ldots$ of elements in $A^{\otimes k+1}$ (notice that $D_i \in \left(A^{\otimes k}\right)^{\otimes}$). Since $a_0, a_1, a_2, \ldots$ all belong to the finite set A, there is an $a \in A$ and an infinite sequence $i_0 < i_1 < \cdots$ such that $a_{i_j} = a$ for all $j \geqslant 0$. Since $\sqsubseteq_1$ is a wqo on $A^{\otimes k}$ by the induction hypothesis, it follows by Lemma 9 that $\sqsubseteq_1^{\otimes}$ is a wqo on $\left(A^{\otimes k}\right)^{\otimes}$. By definition of wqo, there are $i_m < i_n$ with $D_{i_m} \sqsubseteq_1^{\otimes} D_{i_n}$. By definition of $\sqsubseteq_1$ we have that $\langle a_{i_m}, D_{i_m}\rangle \sqsubseteq_1 \langle a_{i_n}, D_{i_n}\rangle$.

We are now ready to show that $\sqsubseteq_2$ is a wqo. Consider an infinite sequence as the one above. Since $\langle a_{i_m}, D_{i_m}\rangle \sqsubseteq_1 \langle a_{i_n}, D_{i_n}\rangle$ and $\sqsubseteq_1 \subseteq \sqsubseteq_2$ it follows that $\langle a_{i_m}, D_{i_m}\rangle \sqsubseteq_2 \langle a_{i_n}, D_{i_n}\rangle$ which completes the proof for wqo of $\sqsubseteq_2$.

Lemma 8 implies that, for extended configurations e_1, e_2, we have that $e_1 \sqsubseteq e_2$ iff $\#e_1 \sqsubseteq_2 \#e_2$. Also, recall that, for $e = \langle G, c\rangle$ the height of G is equal to the order of $\#e$. From this and the fact that $\sqsubseteq_2$ is a wqo on $A^{\otimes k}$ for any given $k \geqslant 1$, we get the following lemma.

Lemma 10. *For any $k \geqslant 1$, the ordering $\sqsubseteq$ is a wqo on $\mathbb{E}^k$.*

7.6 Monotonic Transition Systems

A *monotonic transition system (MTS)* is a tuple $\langle \Gamma, \Gamma_{init}, \sqsubseteq, \longrightarrow, U \rangle$, where

- Γ is a (potentially infinite) set of *configurations*.
- $\Gamma_{init} \subseteq \Gamma$ is a set of *initial* configurations.
- $\sqsubseteq$ is a computable ordering on Γ, i.e., for each $\gamma_1, \gamma_2 \in \Gamma$, we can check whether $\gamma_1 \sqsubseteq \gamma_2$. Furthermore, $\sqsubseteq$ is a wqo.
- $\longrightarrow$ is a binary *transition relation* on Γ. Furthermore, $\longrightarrow$ is monotonic with respect to $\sqsubseteq$, i.e., given configurations $\gamma_1, \gamma_2, \gamma_3$ such that $\gamma_1 \longrightarrow \gamma_2$ and $\gamma_1 \sqsubseteq \gamma_3$, there is a configuration γ_4 such that $\gamma_3 \longrightarrow \gamma_4$ and $\gamma_2 \sqsubseteq \gamma_4$.
- U is defined as the upward closure $\Gamma_1\uparrow$ of a finite set $\Gamma_1 \subseteq \Gamma$, where $\Gamma_1\uparrow = \{\gamma' \in \Gamma \mid \exists \gamma \in \Gamma_1.\ \gamma \sqsubseteq \gamma'\}$.

We use $\xrightarrow{*}$ to denote the reflexive transitive closure of $\longrightarrow$. For sets $\Gamma_1, \Gamma_2 \subseteq \Gamma$, we say that Γ_2 is *reachable* from Γ_1 if there are $\gamma_1 \in \Gamma_1$ and $\gamma_2 \in \Gamma_2$ such that $\gamma_1 \xrightarrow{*} \gamma_2$. In the reachability problem MTS-Reach we are given an MTS $\langle \Gamma, \Gamma_{init}, \sqsubseteq, \longrightarrow, U \rangle$ and are asked the question whether U is reachable from Γ_{init}. The paper [1] gives sufficient conditions for decidability of MTS-Reach as follows. For $\Gamma_1 \subseteq \Gamma$, we define $Pre(\Gamma_1) := \{\gamma \mid \exists \gamma_1 \in \Gamma_1.\ \gamma \longrightarrow \gamma_1\}$. For $\Gamma_1 \subseteq \Gamma$, we say that $M \subseteq \Gamma_1$ is a *minor* set of Γ_1 if

- For each $\gamma_1 \in \Gamma_1$ there is $\gamma_2 \in M$ such that $\gamma_2 \sqsubseteq \gamma_1$.
- If $\gamma_1, \gamma_2 \in M$ and $\gamma_1 \sqsubseteq \gamma_2$ then $\gamma_1 = \gamma_2$.

Since $\sqsubseteq$ is a wqo, it follows that each minor set is finite. However, in general, the same set may have several minor sets. We use min to denote a function which, given $\Gamma_1 \subseteq \Gamma$, returns an arbitrary (but unique) minor set of Γ_1. We use $minpre(\gamma)$ to denote the set $\min(Pre(\{\gamma\}\uparrow))$.

It is shown in [1] that the following conditions are sufficient for decidability of MTS-Reach.

Theorem 11. MTS-Reach *is decidable if for each* $\gamma \in \Gamma$

- *we can check whether* $\gamma \in \Gamma_{init}$.
- *the set* $minpre(\gamma)$ *is finite and computable.*

7.7 From Tree-Bounded-Cover to MTS-Reach

For a natural number $k \geqslant 1$, a process $P = \langle Q, \Sigma, \Delta, q_{init} \rangle$, and a state *target* $\in Q$, we derive an MTS $\langle \Gamma, \Gamma_{init}, \sqsubseteq, \longrightarrow, U \rangle$ such that $\Gamma_{init} \xrightarrow{*} U$ iff there is a Dag G which is an inverted tree with *height* $(G) \leqslant k$ such that *target* is reachable in the Daahn $N = \langle P, G \rangle$.

- Γ is the set $\mathbb{E}^k$.
- Γ_{init} is the set of pairs $\langle G, c_{init} \rangle \in \mathbb{E}^k$ and c_{init} is the initial configuration of the Daahn $\langle G, P \rangle$.
- $\sqsubseteq$ is defined on $\mathbb{E}^k$ as described above. The ordering $\sqsubseteq$ is obviously computable. Well quasi-ordering of $\sqsubseteq$ on Γ is shown in Lemma 10.

- The transition relation $\longrightarrow$ on $\mathbb{E}^k$ is defined as described above. Monotonicity is shown in Lemma 7.
- U is defined as the upward closure of the singleton $\{\langle G_1, c_1\rangle\}$, where $G_1 = \langle\{v\}, \varnothing\rangle$ i.e., G_1 contains a single vertex v and no edges, and furthermore $c_1(v) = \mathit{target}$. Notice that U characterizes all inverted trees that contain at least one vertex labeled with *target*.

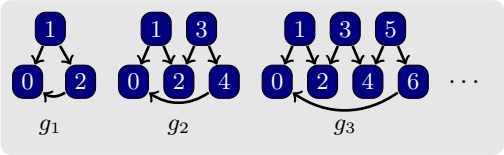
It is trivial to check whether a given configuration is initial (check whether all vertices are labeled with q_{init}). The following lemma states that the induced transition system also satisfies the second sufficient condition for decidability (see Theorem 11).

Lemma 12. *Consider the MTS defined above. Then, for each extended configuration e we can compute $minpre(e)$ as a finite set of extended configurations.*

Lemma 12, together with Theorem 11, proves Theorem 6.

8 Related Work

A fixed, generally small, number of processes has been considered when model checking techniques have been applied to verify ad hoc network protocols [4, 12]. In [11] Saksena et al. define a possibly non-terminating symbolic procedure based on graph transformations to verify routing protocols for Ad Hoc Networks. Delzanno *et al.* showed in [2] that the coverability problem is undecidable in the general case of unbounded, possibly cyclic and directed graphs. In particular, the same authors considered in [3] the bounded-depth subclass of Ad Hoc Networks. Using the induced sub-graph relation on bounded-depth graphs as a symbolic representation within the well quasi-ordered transition systems framework, they proved the decidability of the coverability problem. However, this result cannot be used in the context of directed acyclic ad hoc networks because the induced sub-graph relation is not a well quasi-order in the case of the bounded depth acyclic graphs. In fact, as shown in the figure, the list of directed acyclic labeled graphs of depth 2, $g_1, g_2, g_3, \ldots$ is an infinite sequence of extended configurations of incomparable elements.



9 Conclusions

We have considered parameterized verification of ad hoc networks where the network topology is defined by an acyclic graph. We have considered the coverability problem which, for a given process definition, asks whether there is a graph and a reachable configuration where a process is in a given state. The

coverability problem is used to find violations generated by a fixed set of processes independently from the global configuration. The problem turns out to be undecidable in the general case, but decidable under the restriction that the graph is of bounded depth (where the depth is bounded by a given k). Among possible directions for future work is the study of the impact of richer broadcast mechanisms such as those that allow processes to have local (unbounded) mailboxes, and to consider models augmented by timed and probabilistic transitions in order to allow quantitative reasoning about network behaviors.

References

1. Abdulla, P., Cerans, K., Jonsson, B., Tsay, Y.: General decidability theorems for infinite-state systems. In: LICS'96. pp. 313–321. IEEE Computer Society (1996)
2. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: CONCUR'10. LNCS, vol. 6269. Springer (2010)
3. Delzanno, G., Sangnier, A., Zavattaro, G.: On the power of cliques in the parameterized verification of ad hoc networks. In: FoSSaCS'11. LNCS, vol. 6604, pp. 441–455. Springer (2011)
4. Fehnker, A., van Hoesel, L., Mader, A.: Modelling and verification of the LMAC protocol for wireless sensor networks. In: IFM'07. LNCS, vol. 4591, pp. 253–272. Springer (2007)
5. Higman, G.: Ordering by divisibility in abstract algebras. Proc. London Math. Soc. (3) 2(7), 326–336 (1952)
6. Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., Silva, F.: Directed diffusion for wireless sensor networking. IEEE/ACM Trans. Netw. 11(1), 2–16 (Feb 2003)
7. Levis, P., Patel, N., Culler, D.E., Shenker, S.: Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In: NSDI. pp. 15–28 (2004)
8. Merro, M., Ballardin, F., Sibilio, E.: A timed calculus for wireless systems. Theor. Comput. Sci. 412(47), 6585–6611 (2011)
9. Nanz, S., Hankin, C.: A framework for security analysis of mobile wireless networks. Theor. Comput. Sci. 367(1-2), 203–227 (2006)
10. Prasad, K.V.S.: A calculus of broadcasting systems. Sci. Comput. Program. 25(2-3), 285–327 (1995)
11. Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of Ad Hoc Routing Protocols. In: TACAS'08. LNCS, vol. 4963, pp. 18–32. Springer (2008)
12. Singh, A., Ramakrishnan, C.R., Smolka, S.A.: Query-based model checking of ad hoc network protocols. In: CONCUR'09. LNCS, vol. 5710, pp. 603–619. Springer (2009)
13. Singh, A., Ramakrishnan, C.R., Smolka, S.A.: A process calculus for mobile ad hoc networks. Sci. Comput. Program. 75(6), 440–469 (2010)