Fundamenta Informaticae XX (2018) 1–21
DOI 10.3233/FI-2016-0000
IOS Press

Logic-based Verification of the Distributed Dining Philosophers Protocol

Giorgio Delzanno
DIBRIS
University of Genova
giorgio.delzanno@unige.it

Abstract. We present a logic-based framework for the specification and validation of distributed protocols. Our specification language is a logic-based presentation of update rules for arbitrary graphs. Update rules are specified via conditional rewriting rules defined over a relational language. We focus our attention on unary and binary relations as a way to specify predicates over nodes and edges of a graph. For the considered language, we define assertions that can be applied to specify correctness properties for arbitrary configurations. We apply the language to model the distributed version of the Dining Philosopher Protocol. The protocol is defined for asynchronous processes distributed over a graph with arbitrary topology. We propose then validation methods based on source to source transformations and deductive reasoning. We apply the resulting method to provide a succint correctness proof of the considered case-study.

1. Introduction

Verification of distributed systems with parametric dimension is a challenging task. In this setting protocols are defined on generic configurations of a network, e.g., for an arbitrary number of nodes and arbitrary connection topology. Protocol rules may depend on the current network configuration. For instance, in [28] Namjoshi and Trefler introduced a distributed variant of the dining philosopher protocol in which individual nodes interact asynchronously via shared buffers. Global conditions formulated on the state of visible buffers are used to regulate the access to resources shared between adjacent nodes.

In this paper we present a formalization of distributed protocols with all above mentioned features based on a logic-based presentation of graphs, called GLog. We use graphs in order to model

both complex topologies and asynchronous operations in a more natural way with respect to standard transition systems. Point-to-point links can be obtained by using conditions defined on update rules.

The technical contribution of the paper is as follows. We first define the syntax and formal semantics of the GLog specification language. The language is based on conditional update rules. Conditions are expressed as first order quantified formulas defined over binary predicates without function symbols. Update rules are defined by sets of atomic formulas that specify the ground atoms that have to be removed from the current configuration and those that have to be added to the new one. The interpretation domain of relations is defined over an infinite set of node identifiers. GLog can be viewed as a fragment of richer languages for representing evolving databases like DCDS [19]. Unlike DCDS, designed for updates of relational databases, we are interested here in a minimal fragment of relational logic to reason about evolving graphs. In this setting we can model handshaking between two agents via intermediate steps in which agents first connect to proxies and then, by updating the relations connecting them, start an interaction between them. In our model we admit multiple connections to the same proxy.

Concerning qualitative properties, we focus our attention on a special class of decision problems that can be viewed as parametric versions of reachability problems. More specifically, in order to reason about families of distributed systems we consider reachability problems existentially quantified over initial configurations as in previous work on formal verification of parameterized systems e.g.[11, 12, 13, 4, 2]. In the paper we study minimal fragments of GLog in which the considered decision problem is undecidable. The considered fragments are similar to fragments of multiset rewriting studied in [22, 23].

As a case-study, we apply our language to model the distributed version of the Dining Philosopher Protocol proposed in [29]. The protocol is defined for asynchronous processes distributed over a graph with arbitrary topology. We model the protocol by considering dynamic reconfigurations of the topology. To verify the correctness of our model, we apply a method that combines program transformations and deductive reasoning. Specifically, we first apply source to source transformations guided by commutation (permutation) schemes. Commutation schemes are obtained via a proof theoretic analysis of computations in which pairs are rule applications that can be permuted are used to define a sort of normal form for computations. A similar kind of reasoning is used when reasoning on semantics based on Mazukiewicz traces or partial order reductions. Permutation schemes can be applied to derive meta rules obtained by composing sequences of rule applications of the original protocol. Meta rules can be viewed as specialized protocol rules that embed in their semantics special properties of computations in the considered case-study. Human ingenuity is used here to analyze and infer permutation schemes. We then use deductive reasoning to prove mutual exclusion on the resulting model. Invariants are expressed using parametric formulas, an extension of the assertional language used in GLog. The extended type of assertions are needed in order to specify reachability problems parametric on the number of components. The resulting method is based on proof techniques that are complementary to compositional reasoning [29] and symbolic backward exploration [18].

2. GLog

In this section we will define a logic-based presentation of evolving graphs called GLog, GLog generalizes the BLog language in [8]. GLog formulas are based on a simple relational calculus that can be used to express updates of sets of ground atoms. A set of ground atoms can be interpreted as the current state or configuration of the system we are modeling. Update rules contain a formula working as a condition and deletion and addition sets that specify ground atoms to be deleted and added to the current state.

More formally, let P be a finite set of names of (unary and binary) predicate names, $\mathcal{N}$ a denumerable set of node identifiers equipped with a total order $<$, V be a denumerable set of variables. Predicates in P are used to model current configurations. In addition to predicates in P, we interpret the binary relation lt as the total order $<$ in our model. Our logic has no function symbols but can be instantiated with elements from $\mathcal{N}$. An atomic formula is either a formula $p(x)$, $lt(x, y)$ or $p(x, y)$, where $p \in P$, $x, y \in V \cup \mathcal{N}$ A ground atom is a either a $p(n)$, $lt(n, m)$, or $p(n, m)$, where $n, m \in \mathcal{N}$. A literal is either an atomic formula or the negation $\neg A$ of an atomic formula A. A formula is a first order formula built on literals, namely, any literal is a formula, conjunctions, disjunctions, universally and existentially quantified formulas are still formulas. Multiple occurrences of the same variable implicitly model equality constraints. The set of free variables of a formula F, namely $FV(F)$, is the minimal set satisfying

- $FV(p(x, y)) = \{x, y\}$,
- $FV(A \vee B) = FV(A) \cup FV(B)$,
- $FV(A \wedge B) = FV(A) \cap FV(B)$,
- $FV(\neg A) = FV(A)$,
- $FV(\forall v.A) = FV(A) \setminus \{v\}$,
- $FV(\exists v.A) = FV(A) \setminus \{v\}$.

Given $S = \{F_1, \ldots, F_n\}$, we define $FV(S) = FV(F_1) \cup \ldots \cup FV(F_n)$. Quantified formulas we will be used as application conditions of rules.

2.1. Configurations, Interpretations and Update Rules

As mentioned before a set of ground atoms will be used to model a configuration. Formally, a configuration is a finite set Δ of ground atomic formulas with predicates in P. A configuration implicitly defines a graph in which directed edges are represented by atomic formulas whose predicate name acts as edge label. Configurations can also be viewed as models in which to evaluate a conditions.

An interpretation is a mapping σ from V to $\mathcal{N}$. We use here a fixed interpretation of variables. The interpretation domain however consists of a denumerable set of node identifiers. For a formula F we use $F\sigma$ as an abbreviation for $\hat{\sigma}(F)$, where $\hat{\sigma}$ is the natural extension of σ to terms. For a set $S = \{A_1, \ldots, A_n\}$, we use $S\sigma$ to denote the set $\{A_1\sigma, \ldots, A_n\sigma\}$.

Update rules consists of conditions defined by quantified formulas with no function symbols, a deletion and an addition set. The deletion (resp. addition) set defines the set of ground atoms that have to be cancelled from (resp. added to) the current configuration. A rule has the following form $\langle C, D, A\rangle$, where C is a quantified formula, D and A are two sets of atomic formulas with variables in V and predicates in P, and such that $FV(A) \cup FV(D) \subseteq FV(C)$.

To fix an operational semantics for our language we need a support for the interpretation of relations and variables.

We use $\Delta \models A$ to define the satisfiability relation of a quantified formula A s.t. $FV(A) = \emptyset$. Let $A[n/X]$ denote the formula obtained by replacing each free occurrence of X with n. The relation is defined by induction as follows.

- $\Delta \models p(n)$, if $p(n) \in \Delta$;
- $\Delta \models lt(n, m)$, if $n < m$;
- $\Delta \models p(n, m)$ for $p \in P$, if $p(n, m) \in \Delta$;
- $\Delta \models A \wedge B$, if $\Delta \models A$ and $\Delta \models B$;
- $\Delta \models \neg A$, if $\Delta \not\models A$;
- $\Delta \models \forall X.A$, if $\Delta \models A[n/X]$ for each $n \in \mathcal{N}$;
- $\Delta \models \exists X.A$, if $\Delta \models A[n/X]$ for some $n \in \mathcal{N}$.

Given a configuration Δ, we say that the quantified formula A is satisfied in Δ, if there exists an interpretation σ s.t. $A\sigma$ is satisfiable.

In order to apply a rule $\langle C, D, A\rangle$ to Δ, there must be an interpretation σ that satisfies the quantified formula C. The same interpretation σ is then applied to the atomic formulas in D and A. The resulting sets of atoms, say D' and A' respectively, are deleted from and added to Δ, respectively.

2.2. Transition System

A protocol $\mathcal{P}$ is a set of rules. The operational semantics of $\mathcal{P}$ is given by a transition system $T_{\mathcal{P}} = \langle \mathcal{C}, \rightarrow\rangle$, where $\mathcal{C}$ is the set of possible configurations, i.e., finite subsets of ground atoms with predicates in P, and $\rightarrow \subseteq \mathcal{C} \times \mathcal{C}$ is a relation defined as follows.

For $\Delta, \Delta' \in \mathcal{C}$ and a rule $\langle C, D, A\rangle \in \mathcal{P}$, $\Delta \rightarrow \Delta'$ if there exists σ s.t. $\Delta \models C\sigma$ and $\Delta' = (\Delta \setminus D\sigma) \cup A\sigma$. A computation is a sequence of configurations $\Delta_0\Delta_1 \ldots$ s.t. $\Delta_i \rightarrow \Delta_{i+1}$ for $i \geq 0$.

We use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$. In a single step of the operational semantics a rule is evaluated in the current configuration by taking a sort of closed-world assumption, i.e., ground atomic formulas that do not occur in a configuration are evaluated to false. Furthermore, ground atomic formulas that are not deleted are transferred from the current to the successor configuration. The latter property can be viewed then as a sort of frame axiom. It is important to notice that, in general, a configuration Δ has several possible successors. Indeed, depending of the chosen interpretation of free variables the same rule can be applied to different subsets of ground atoms contained in the same configuration. Furthermore, the choice of the rules to be applied at a given step is non-deterministic.

2.3. Protocol Example

As an example, we consider possible application of GLog to the specification of distributed protocols. The key ingredient of the specification language is the combination of complex conditions and update rules to reason on graphs in which predicates can be viewed as labels of links between agents and communication buffers. We have shown that we can also add labels to individual agents and buffers, e.g., to represent their current state. Update rules can be used to dynamically reconfigure the graph, i.e., change labels, topology and add or delete agents. The separation between agents and buffers is convenient to model asynchronous communication. For instance, let us consider a protocol in which two agents need to establish a connection via a shared buffer.

- An agent n_1 of type A connects to a buffer e_1 in idle state (the buffer is free) and sets the state of the buffer to $ready$.
- An agent n_2 of type B connects to e_1 in state $ready$ and changes the state to $readyack$.
- Agent n_1 sends message m by changing the state of e_1 to msg_m.
- Agent n_2 receives message m and updates the state of the channel to $readyack$ for further communications.

The protocol can be specified as follows. We use unary predicates to associate states to edges. $send$ messages are non-deterministically generated.

R	C	D	A
1	$idle(B) \wedge \neg req(A,B)$	$\{idle(B)\}$	$\{ready(B), req(A,B)\}$
2	$ready(B) \wedge \neg rec(A,B)$	$\{ready(B)\}$	$\{readyack(B), rec(A,B)\}$
3	$true$	$\{\}$	$\{send(A,B,M)\}$
4	$readyack(B) \wedge send(A,B)$	$\{readyack(B), send(A,B)\}$	$\{msg(B,M)\}$
5	$msg(B,M) \wedge rec(A,B)$	$\{msg(B,M), rec(A,B)\}$	$\{idle(B)\}$

An initial configuration has the form $idle(b_1), \ldots, idle(b_k)$, where $b_i < b_j$for $i \neq j$, $i, j : 1, \ldots, k$. For the sake of simplicity, we do not model the state of agents but only their capabilities (req, rec, send). In rule 1 a buffer B is locked by a non-deterministically generated request $req(A, B)$ from sender agent A (a variable). In rule 2 a buffer B is locked by a non-deterministically generated request $rec(A, B)$ from receiver agent A (a variable). Rule 3 nondeterministically generates a send action from agent A. Rule 4 synchronizes a send action from agent A with a buffer locked by the same agent. The (non deterministically generated) message M is stored in the buffer. Rule 5 synchronizes and consumes a message in the buffer with the receiver agent, releasing the buffer.

The model provides other form of interactions. For instance, we can model ordered buffers by forming lists of messages attached to a given edge as in the representation of the tape of the Turing machine.

We can also model synchronous communication as in the following example

C	$link(A,B) \wedge s_1(A) \wedge link(E,B) \wedge s_2(E)$
D	$\{s_1(A), link(A,B), link(E,B), s_2(E)\}$
A	$\{link(A,B), s'_1(A), link(E,B), s'_2(E)\}$

Here $s(A)$ and $s'(A)$ denote agent A resp. in state s and s', $s_1(E)$ and $s'_1(E)$ denote agent E resp. in state s_1 and s'_1, and $link(A,B)$ and $link(Y,B)$ denote links to a common buffer B.

3. Decision Problem

We consider decision problems that generalize the standard notion of reachability between configurations. The key point is to reason about an infinite set of initial configurations in order to prove properties for protocol instances with an arbitrary number of nodes. For a set S of configurations, we first define the $Post$ and Pre operators as follows $Post(S) = \{\Delta' \mid \exists \Delta \in S,\ \Delta \rightarrow \Delta'\}$ and $Pre(S) = \{\Delta' \mid \exists \Delta \in S,\ \Delta' \rightarrow \Delta\}$. We use $Post^*(S)$ (resp. $Pre^*(S)$) to denote the reflexive-transitive closure of $Post$ (resp. Pre).

We now introduce the $\exists$-reachability problem defined as follows.

Definition 3.1. Given a protocol $\mathcal{P}$, a set of target configurations T and a possibly infinite set of initial configurations I, $\exists$-reachability is satisfied for $\mathcal{P}$, I and T, written $\exists Reach(\mathcal{P}, I, T)$, if there exists $\Delta \in T$ and a configuration Δ_1 s.t. $\Delta_1 \in Post^*(I)$ and $\Delta \subseteq \Delta_1$.

By expanding the definition of $Post^*$, $\exists Reach(\mathcal{P}, I, T)$ holds if there exists a configuration $\Delta_0 \in I$ s.t. $\Delta_0 \rightarrow^* \Delta_1$ and $\Delta \subseteq \Delta_1$ for some $\Delta \in T$. The target T can be interpreted as a pattern to match or avoid in computations starting from initial configurations. If the set I consists of configurations consisting of an arbitrary, finite number of components than $\exists$-reachability formally describes a parameterized verification decision problem for specifications given in GLog.

3.1. Undecidability

The $\exists$-Reachability problem turns out to be undecidable [8]. A proof can be constructed by reducing the halting problem for Turing machines to $\exists$-reachability. The idea of the construction is as follows.

Let $M = \langle \Sigma, Q, \delta, q_0, F \rangle$ be a deterministic Turing machine in which Q is the set of control states, Σ is the tape alphabet that includes the special blank symbol B, $F \subseteq Q$ is the set of final states, q_0 is the initial state, and $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$ is the transition relation. We consider the halting problem from an initial state with empty tape. To encode a configuration of M, we use here three types of relations:

- The atomic formula $q(n_1)$ represents the state of the head pointing to cell n_1.
- The atomic formula $\ell(n_1)$ represents a cell n_1 containing symbol ℓ.
- The atomic formula $next(n_1, n_2)$ is used to link cell n_1 to its successor cell n_2.

To simulate a transition $\delta(q_1, a) = \langle q_2, b, R \rangle$ we use the rules defined as follows

R	C	D	A
1	$q_1(X_1) \wedge a(X_1) \wedge$ $next(X_1, X_2) \wedge \ell(X_2)$	$\{q_1(X_1), a(X_1)\}$	$A_1 = \{q_2(X_2), b(X_1)\}$
2	$q_1(X_1) \wedge a(X_1) \wedge lt(X_1, X_2) \wedge$ $\neg \exists Z.next(X_1, Z)$	$\{q_1(X_1), a(X_1)\}$	$\{q_2(X_2), b(X_1), blank(X_2)\}$

The former rule updates the relation associated to the encoding of the cell pointed by the head assuming that there exists another cell to its right. The latter rule generates a representation of a blank cell by selecting an unused identifier. To simulate a transition $\delta(q_1, a) = \langle q_2, b, L \rangle$ we use the rules and where

R	C	D	A
3	$q_1(X_1) \wedge a(X_1) \wedge next(X_2, X_1)$ $\wedge \ell(X_2)$	$\{q_1(X_1), a(X_1)\}$	$\{q_2(X_2), b(X_1)\}$
4	$q_1(X_1) \wedge a(X_1) \wedge$ $lt(X_2, X_1) \wedge \neg \exists Z.next(Z, X_1)$	$\{q_1(X_1), a(X_1)\}$	$\{q_2(X_2), b(X_1), blank(X_2)\}$

The former rule updates the relation associated to the encoding of the cell pointed by the head assuming that there exists another cell to its left. The latter rule generates, when needed, a representation of a blank cell.

The set of initial configurations I consists of configurations with a single occurrence of atoms $q_0(n)$ and $blank(n)$ for some $n \in \mathcal{N}$. Starting from $\Delta \in I$ our encoding produces configurations of a M. Let us indicate with $\overline{\Delta}$ the configuration of the Turing machine represented by Δ. Namely, if $\Delta = \{a_1(n_1), next(n_1, n_2), a_2(n_2), \ldots, next(n_{i-1}, n_i), q(n_i), a(n_i), next(n_i, n_{i+1}), \ldots, a_k(n_k)\}$ then $\overline{\Delta} = a_1 \ldots a_{j-1} q_i a_j \ldots a_k$.

The following properties then holds.

Proposition 3.2. Let $\mathcal{P}_M$ be the encoding of the Turing machine M and $\gamma_0 = q_0 B$ be the initial configuration of M. γ_1 is reachable from γ_0 if and only if there exists $\Delta_0 \in I$ and Δ_1 s.t. $\overline{\Delta_0} = \gamma_0$, $\overline{\Delta_1} = \gamma_1$, and $\Delta_0 \rightarrow^* \Delta_1$.

Proof:
By construction, the application of a rule produced by the encoding preserve the well-formedness of the encoding of configurations. We prove it by induction on the length of a computation. Let us consider the initial configuration $\Delta = \{q_0(n), blank(n)\}$ s.t. $\overline{\Delta_0} = \gamma_0$. By applying C_2 we obtain a configuration

$$\Delta = \{q_1(n_1), blank(n_1), a(n), next(n, n_1)\} \;\; n < n_1$$

that represents configuration aq_1B. C_3 produces a similar effect by adding a blank to the left of the head positions. Now let

$$\Delta = \{a_1(n_1), next(n_1, n_2), a_2(n_2), \ldots, q(n_i), a(n_i), next(n_i, n_{i+1}), \ldots, a_k(n_k)\}$$

be a configuration obtain after m steps s.t. $\overline{\Delta} = a_1 a_2 \dots a_k$. If $i < k$ by applying rule C_1, we obtain

$$\Delta' = \{a_1(n_1), next(n_1, n_2), a_2(n_2), \dots, a_k(n_k), \dots, a'(n_i), next(n_i, n_{i+1}), q'(n_{i+1}), \dots, a_k(n_k)\}$$

and $\overline{\Delta'} = a_1 a_2 \dots a' q' a_{i+1} \dots a_k$. If $i = k$ by applying rule C_2, we obtain

$$\Delta = \{a_1(n_1), next(n_1, n_2), a_2(n_2), \dots, a'(n_i), next(n_i, n'), q'(n'), blank(n')\}\ n_i < n'$$

and $\overline{\Delta'} = a_1 a_2 \dots a' q' a_{i+1}$. A similar case analysis applies for rules that move the tape head to the left.

Viceversa, if γ_1 is reachable from γ_0, then, by construction, we can build a computation from $\Delta = \{q_0(n), blank(n)\}$ that mimics each step of the machine M. The proof is again for induction on the length of the computation in M with a case analysis similar to the if case. □

The following property then holds.

Theorem 3.3. Let $\mathcal{P}_M$ be the encoding of the Turing machine M. The Halting problem for M can be reduced to $\exists$-Reachability in $\mathcal{P}_M$ by taking the formula $\Phi = \bigvee_{q \in F} q(X, Y)$ as representation of sub-configurations denoting a final state.

3.2. Fragments of GLog

We consider here a restricted form of update rules, namely we only consider relabeling rules.

Definition 3.4. A relabeling rule is an update rule of the form $\langle C, D, A \rangle$, where

- $C = D = \{p_1(X_1, Y_1), \dots, p_k(X_k, Y_k)\}$
- $A = \{q_1(X_1, Y_1), \dots, q_k(X_k, Y_k)\}$

where $p_i, q_i \in P$ for $i : 1, \dots, k$.

Observe that D is used both as enabling condition and deletion set so as to ensure the existence of all elements to be deleted. Free variables occurring in D and A are implicitly existentially quantified. For instance, consider the rule $\langle D, D, A \rangle$ such that $D = \{p(X, Y)\}$ and $A = \{q(X, Y)\}$. The previous rule non-deterministically selects a link with label p and updates its label to q. Relabeling rules only change labels of existing links. We focus our attention on relabeling rules since our assertional language cannot represent arbitrary formulas as those allowed in conditions. Relabeling however induces very expressive transition systems. To clarify this point, we first say that a set of configurations S is regular if there exists a set of rules with associated successor operator $Post$ and a finite set I of configurations such that $S = Post^*(I)$. In other words S can be generated by a finite set of configurations by applying GLog rules. The following property holds for regular set of initial configurations.

Proposition 3.5. The $\exists$-reachability problem is undecidable for GLog specifications consisting of relabeling rules and regular set of initial configurations.

Proof:
We present here a sketch of the proof. We first notice that edge addition and deletion can be simulated by introducing a special predicate name (i.e. edge label) ϵ. To simulate edge creation, we can use relabeling rules that update ϵ-edges. For instance, the rule $D = \{\epsilon(X,Y)\}, A = \{p(X,Y)\}$ non-deterministically introduces an occurrence of a p-edge. Similar constructions have been proposed for graph rewriting with addition/deletion of edges or with relabeling in [3]. Undecidability of $\exists$-reachability depends from the assumptions on the shape of initial states.

Let us consider initial states consisting of (directed) paths (i.e. chains of atomic formulas) of arbitrary length with an initial label q_0 followed by ϵ labels only, then we can use such a configuration as initial structure on which to run a simulation of a Turing powerful model such a two counter machine. Every path corresponds to the maximum amount of memory needed during the execution of a machine. The initial state q_0 can then be transformed into a pointer to visit each element (an edge) of the memory and to modify its contents (e.g. flip it from one to zero or from zero to one). To represent the value k for counter c with use k occurrences of label c along the path.

More specifically, if the transition from q_1 to q_2 increments counter c we use the following set of rules:

R	D	A
1	$\{q_1(X,Y), \epsilon(Y,Z)\}$	$\{q_2(X,Y), c(Y,Z)\}$
2	$\{q_1(X,Y), c(Y,Z)\}$	$\{q_1^1(X,Y), c^1(Y,Z)\}$
3	$\{q_1(X,Y), d(Y,Z)\}$	$\{q_1^1(X,Y), d^1(Y,Z)\}$
4	$\{c^1(X,Y), c(Y,Z)\}$	$\{c^1(X,Y), c^1(Y,Z)\}$
5	$\{c^1(X,Y), \epsilon(Y,Z)\}$	$\{c^2(X,Y), c(Y,Z)\}$
6	$\{d^1(X,Y), \epsilon(Y,Z)\}$	$\{d^2(X,Y), c(Y,Z)\}$
7	$\{d^1(X,Y), d(Y,Z)\}$	$\{d^1(X,Y), d^1(Y,Z)\}$
8	$\{d^1(X,Y), c(Y,Z)\}$	$\{d^1(X,Y), c^1(Y,Z)\}$
9	$\{c^1(X,Y), c^2(Y,Z)\}$	$\{c^2(X,Y), c(Y,Z)\}$
10	$\{c^1(X,Y), d^2(Y,Z)\}$	$\{c^2(X,Y), d(Y,Z)\}$
11	$\{d^1(X,Y), d^2(Y,Z)\}$	$\{d^2(X,Y), d(Y,Z)\}$
12	$\{c^1(X,Y), d^2(Y,Z)\}$	$\{c^2(X,Y), d(Y,Z)\}$
13	$\{q_1^1(X,Y), c^2(Y,Z)\}$	$\{q_2(X,Y), c(Y,Z)\}$
14	$\{q_1^1(X,Y), d^2(Y,Z)\}$	$\{q_2(X,Y), d(Y,Z)\}$

The rules just scan the chain passing through c and d labels until the first ϵ label is found. During the scan each label is marked to prepare the second phase of the simulation. When the ϵ label has been found, it is replaced with c and then the chain is traversed back in order to restore the original c and d labels and to move the control state to q_2. Decrement of counter d is handled in a simmetric way.

If the transition from q_1 to q_2 decrement counter c we use the following set of rules:

R	D	A
$1b$	$\{q_1(X,Y), c(Y,Z)\}$	$\{q_2(X,Y), \epsilon(Y,Z)\}$
$2b$	$\{q_1(X,Y), d(Y,Z)\}$	$\{q_1^1(X,Y), d^3(Y,Z)\}$
$3b$	$\{d^3(X,Y), d(Y,Z)\}$	$\{d^3(X,Y), d^3(Y,Z)\}$
$4b$	$\{d^3(X,Y), c(Y,Z)\}$	$\{d^4(X,Y), \epsilon(Y,Z)\}$
$5b$	$\{d^3(X,Y), d^4(Y,Z)\}$	$\{d^4(X,Y), d(Y,Z)\}$
$6b$	$\{q_1^1(X,Y), d^4(Y,Z)\}$	$\{q_2(X,Y), d(Y,Z)\}$

The rules just scan the chain passing through d labels until the first c label is found. During the scan each label is marked to prepare the second phase of the simulation. When the c label has been found, it is replaced with ϵ and then the chain is traversed back in order to restore the original d labels and to move the control state to q_2. Decrement of counter d is symmetric.

Similarly, zero test on counter c is implemented by the rules

R	D	A
1c	$\{q_1(X,Y), \epsilon(Y,Z)\}$	$\{q_2(X,Y), \epsilon(Y,Z)\}$
2c	$\{q_1(X,Y), d(Y,Z)\}$	$\{q_1^1(X,Y), d^5(Y,Z)\}$
3c	$\{d^5(X,Y), d(Y,Z)\}$	$\{d^5(X,Y), d^5(Y,Z)\}$
4c	$\{d^5(X,Y), \epsilon(Y,Z)\}$	$\{d^6(X,Y), \epsilon(Y,Z)\}$
5c	$\{d^5(X,Y), d^6(Y,Z)\}$	$\{d^6(X,Y), d(Y,Z)\}$
6c	$\{q_1^1(X,Y), d^6(Y,Z)\}$	$\{q_2(X,Y), d(Y,Z)\}$

The rules scan the chain passing through all d labels in search of the first ϵ. The test fails if there are c labels (i.e. counter c is greater than one).

Rules of non-zero test are implemented in a similar way.

The simulation is successful only for paths with the necessary amount of memory (i.e. length). The halting problem for such a program can be formulated as an existential reachability problem via an assertion $\exists X, Y.link(halt, X, Y)$ used to single out the halting control state of the program. □

The freedom in fixing the shape of the initial set of configurations is central here in order to give enough power to relabeling rules. Indeed, if the initial configurations are arbitrary graphs, e.g., with ϵ labels, then relabeling is not expressive enough to make existential reachability undecidable.

Proposition 3.6. $\exists$-reachability for relabeling rules is decidable if the set of initial configurations consists of arbitrary fully connected graphs with ϵ-transition.

Proof:
The difficulty in building a perfect simulation of a (Turing or counter) machine when starting from arbitrary graphs with epsilon transitions is due to the fact that relabeling rules can non-deterministically

be applied at any position. For every configuration in which a relabeling is applied to a given link, we can find another configuration with additional links attached to the same nodes, involved in the first application of the rule, in which the same relabeling can be applied several times. Since the additional links can be adjacent to the original ones, it is not possible to use chains of predicates to define lists of cells or other regular structures. In other words, the effect of a relabeling rule is only that of enabling other possible rule applications. The same effect can be obtained by considering rewriting rules operating on a finite set of labels (predicate symbols). As a side effect there is no more difference between links and paths.

$\exists$-reachability can then be reformulated as a reachability problem starting from the singleton containing ϵ and with the set of labels occurring in the target assertion as final configuration. Since we only consider finitely many predicate symbols, this kind of reachability problem becomes decidable. Similar reductions have been proposed for coverability in graph rewriting with relabeling rules, and in broadcast protocols with non deterministic reconfigurations of links [10]. □

The previous result shows that relabeling rules defined over finite alphabets are not very expressive.

4. Distributed Dining Philosophers

We consider here a distributed version of the dining philosopher mutual exclusion problem presented in [28]. Agents are distributed on an arbitrary graph and communicate asynchronously via point-to-point channels. Channels are viewed as buffers with state. Distributed Dining Philosophers (DDP) is defined as follows. The goal is to ensure that agents can access a resource shared in common with their neighbors in mutual exclusion. The protocol from the perspective a single agent consists of the following steps:

- Initially, all agents are in *idle* state.
- When an agent A wants to get a resource, A has to acquire the control of each buffer shared with his/her neighbors.
- To acquire a channel, A marks the channel with its identifier. If the channel is already marked, A has to wait.
- A acquires the resources when all channels shared with neighbors are marked with his/her identifier.
- To release a resource, A first resets each buffer. When all buffers are reset, A moves back to idle state.

In a statically defined topology, agent A gets access to a resource when all neighbors are either idle or are waiting for acquiring some channel. Communication between two neighbors is asynchronous. Indeed, they interact by reading and writing on the shared channel. The protocol should guarantee that two agents that share the same channel cannot acquire and use a resource simultaneosly. The protocol should be robust under dynamic reconfigurations of the network.

4.1. Formal Specification of DDP

In this section we present a formal specification of the DDP protocol. Network configurations are expressed as GLog configurations. The dynamics in a protocol interaction is expressed via a finite set of update rules. We use a predicate $link$ to represent connections from an agent to a possibly shared buffer. To model dynamic reconfigurations, we can non-deterministically add and remove $link$ predicates between pairs of agents and buffers. We model buffers with states using unary predicates. Asynchronous communication is modeled as in the previous example, i.e., agents interact only via a common buffer. Communication between two agents is not atomic. Instead of modeling identifiers and buffers with data, we introduce a special relation own that is used to model ownership of a given buffer to which a agent is linked. Ownership is normed in the same way as the labeling of buffers in the original protocol, i.e., an agent can acquire ownership only if the buffer is not owned by other agents. We model this behavior using the following predicates and rules (rules have the form (C_i, D_i, A_i) for $i : 1, \ldots, 6$):

R	C	D	A
$link$	$\neg link(X, E)$	$\emptyset$	$\{link(X, E)\}$
$unlink$	$link(X, E)$	$\{link(X, E)\}$	$\emptyset$
$getE$	$link(X, E) \wedge \forall Z.\neg own(Z, E)$	$\emptyset$	$\{own(X, E)\}$
$relE$	$\{own(X, E)\}$	$\{own(X, E)\}$	$\emptyset$
$acquire$	$idle(X) \wedge \forall E.(link(X, E) \supset own(X, E))$	$\{idle(X)\}$	$\{busy(X)\}$
$release$	$\{busy(X)\}$	$\{busy(X)\}$	$\{idle(X)\}$

An initial state configuration has the following form $idle(n_1), \ldots, idle(n_k)$, where $n_i \neq n_j$ for $i \neq j$, $i, j : 1, \ldots, k$ and $k \geq 1$.

5. Correctness Proof

Correctness Mutual exclusion for the considered protocol can be formulated in its more general form, i.e., for any number of agents, as the negation of an $\exists$-reachability problem. More specifically, let Φ be the formula

$$\exists N, M, G.busy(N) \wedge busy(M) \wedge link(N, G) \wedge link(M, G)$$

that denotes a configurations in which two agents share the same channel. $\exists Reach(DDP, I, \Phi)$ holds if and only if mutual exclusion does not hold for DDP.

Taking DDP as main example, in the rest of the paper we will present an incomplete verification methodology for the $\exists$-Reachability problem. Our approach follows the principles of deductive verification, i.e., prove that a given assertion is an invariant under application of the protocol rules. The method is combined with protocol transformations guided by permutation and deletion schemes. Transformations produce simpler rules schemes that, combined with the above mentioned proof method, yield correctness proofs of very simple form.

5.1. Code-to-code Transformations

We start our analysis by introducing a sort of canonical form for computations in $\mathcal{P}_{ddp}$ obtained via permutation and deletion schemes that we can be used to synthesize derived rules. We first consider deletion and permutation properties of rule applications within a given computation. We focus our attention on $link$ and $unlink$ rules, i.e., dynamic reconfigurations of the graph topology.

5.1.1. Permutation of $link/unlink$

Let $\theta = \Delta_0\Delta_1\Delta_2$ be a computation in $\mathcal{P}_{ddp}$ and let r_1, r_2 be the transitions applied in each step.

- We first observe that if r_2 is an instance of the $unlink$ rule applied to n_1, e_1, and r_1 is an instance of rule $link$ applied to the same pair, they can be eliminated since $\Delta_0 = \Delta_2$.

- If r_1 is an instance of the $getE$ rule applied to agent n_1, e_1, and r_2 is an application of $relE$ applied to the same pair, then they can be eliminated since $\Delta_0 = \Delta_2$.

- We now observe that $unlink$ can be permuted with every other rule applied to different pairs to its left in a computation. Namely, if r_2 is an instance of the $unlink$ rule applied to n_1, e_1, and r_1 is a rule that is not applied to n_1, e_1, then we can permute the application of the two rules and obtain a new computation leading to the same configuration.

- If r_1 is an instance of the $link$ rule applied to n_1, e_1, and r_2 is an application of any other rule on a different pair, then we can permute the applications of the two rules and obtain a new computation leading to the same configuration.

We can now reason on the previous properties in order to consider sets of equivalent computations and infer derived blocks of rules and new permutation rules. More specifically, given a computation θ we can obtain an equivalent, w.r.t. our correctness criteria, computation θ' by applying the following steps:

- We can eliminate all applications of reconfigurations, i.e., $link$ and $unlink$, performed on the same pair n, e, if in between their occurrences in a computation there are no occurrences of instances of $getE$ on the same pair. The property can be obtained by repeatedly applying permutation rules so as to push applications of $link$ towards the first occurrence of $unlink$ and then apply the corresponding deletion rules.

- We can eliminate all applications of acquire $getE$ and $relE$ on the same pair n, e, if there are no occurrences of $acquire$ involving n in between. Again, the property can be obtained by repeatedly applying permutation rules so as to push applications of $relE$ towards the first occurrence of $getE$ to its left.

- We can push the application of $link$ rules close to the first occurrence of a corresponding $getE$ rule to its left.

Following from the previous properties, from any computation we can extract a subcomputation in which occurrences of $relE$ occur only after occurrences of $acquire$ on the same pair. Similarly, we can push occurrences of $unlink$ so as to occur only after occurrences of $acquire$. In other words we can derive rules obtained by combining $link$ and $reqE$. The resulting pattern, $link\_getE$, is used to simultaneously link an agent to an edge and acquire its ownership. Similarly, since computations in which $relE$ and $unlink$ on specific pairs occur after $acquire$, we can use permutations in order to cluster all occurrences of patterns $relE\_unlink$ occurring before an $acquire$. In other words, we can use permutations of patterns $link\_getE$ with other rules or patterns operating on different agents, in order to push all their occurrences close to the first acquire on their right involving the same agent. The above properties lead to patterns of the form $(link\_getE)^*acquire$. We now observe that the pattern can be used by agent n to simultaneously acquire the ownership of a given set of edges $e_1, \ldots, e_n$, and update the local state to $busy$.

We can reason on permutations of $unlink$, $relE$ and $release$ in a similar way, i.e., apply permutations to cluster the sequence of applications of these rules involving the same agent after an acquire. The resulting sequence can be represented by the pattern $(relE\_unlink)^*release$. In other words in the resulting patterns the $link$ relation is updated in parallel with the own relation. The patterns identified in the previous section cannot be represented via a single rule in our specification language, since they may involve an arbitrary number of edges. To express our patterns, we will use families of updates rules and use them to reason about the correctness of our specification

5.1.2. Link-Request Pattern

To express the pattern $(link\_getE)^*acquire$, we can use a family $R = \{r_k\}_{k\geq 0}$ of rules, where $r_k = (C_k, D_k, A_k)$ is defined as:

- $C_k = idle(X) \wedge (\bigwedge_{i=1}^{k} \neg link(X, E_i)) \wedge (\forall Z. \bigwedge_{i=1}^{k} \neg own(Z, E_i))$
- $D_k = \{idle(X)\}$
- $A_k = \{busy(X), link(X, E_1), own(X, E_1), \ldots, link(X, E_k), own(X, E_k)\}$

The rule associates agent X to an arbitrary subset of edges. The association is defined via the ownership predicate.

5.2. Release-Unlink Pattern

The pattern $(relE\_unlink)^*release$ is expressed by an infinite family $S = \{s_k\}_{k\geq 0}$ of rules, where rule $s_k = (C_k, D_k, A_k)$ is defined as:

- $C_k = busy(X) \wedge (\bigwedge_{i=1}^{k} link(X, E_i) \wedge own(X, E_i))$
- $D_k = \{busy(X), link(X, E_1), own(X, E_1), \ldots, link(X, E_k), own(X, E_k)\}$
- $A_k = \{idle(X)\}$

This rule can be applied to a subset of all edges connected to a given agent. This corresponds to a sequence of applications of $release$ and $relE$.

We will refer to the specification containing the two families of rules R and S as DDP$^{\#}$. The operational semantics can be extended in such a way that transitions are applied by selecting an instantiation from one of the rules in $R \cup S$. The following proposition then holds.

Proposition 5.1. If $\exists Reach(DDP^{\#}, I, \Phi)$ does not hold, then $\exists Reach(DDP, I, \Phi)$ does not hold.

Proof:
We first observe that, by construction, every computation in DDP can be transformed into a computation in DDP$^{\#}$ by exploiting the permutation and deletion schemes described at the beginning of this section. The computation is obtained by reordering the application of transitions and by removing unnecessary attempts of linking to nodes that cannot be owned.

Computations in the abstract protocol can be decomposed into simpler steps of the original protocol. Furthermore, an arbitrary number of link/unlink steps can be reintroduced in the computation. However, the resulting intermediate configurations correspond to attempts that cannot bring an agent to the busy state. Thus, it is not possible that intermediate configurations satisfy Φ while configurations restricted to application of rules in DDP$^{\#}$ do not satisfy Φ.

We now observe that DDP$^{\#}$ contains computations that do not correspond to computations in DDP. Indeed, while patterns in R can be decomposed into sequence of rule applications taken from DDP, there are patterns in S that do not correspond to computations in DDP. This may happen when an instance for a given k of a rule in S is applied to a strict subset of the existing set of links owned by an agent. In this case the some of the edges are not released by the agent. Since the agent moves to state $idle$ those edges will remain blocked forever, making the corresponding buffer unusable in successive computations. This corresponds to a sort of loss transition in which some of the buffers move to a sink state. This overapproximation however is conservative w.r.t. our correcteness proof. If bad configurations cannot be reached in DDP$^{\#}$, then they will not be reached in DDP. □

5.3. Deductive Proof

We now move to a formal correctness proof of our DDP specification by reasoning on DDP$^{\#}$. Our goal is to prove that the formula

$$\Psi = \neg\exists N, M, G.busy(N) \wedge busy(M) \wedge link(N, G) \wedge link(M, G)$$

holds in the initial states and that it is an invariant for DDP$^{\#}$. This will give us a proof that existential reachability does not hold for Ψ in DDP$^{\#}$. By Prop. 5.1, we will obtain a correctness argument that can also be applied to DDP.

To prove the property, we will strenghten the invariant proving that

$$\Psi' = \Upsilon \wedge \Psi$$

where

$$\Upsilon = (\forall Z, E.link(Z, E) \supset own(Z, E))$$

is still an invariant of our model. By definition Ψ' holds in any initial configuration $\Delta_{m,n}$, since initial configurations only contain agents in state $idle$. The key point is to show that Ψ' is preserved by applications of rule r_k and s_k for any $k \geq 0$.

Consider a configuration Δ with cardinality n and assume that $\Delta \models \Psi'$. Now consider r_k s.t. $k \leq n$. If r_k can be applied to Δ, there exists an interpretation σ s.t. $\Delta \models C_k\sigma$ where

$$C_k = (idle(X) \wedge \bigwedge_{i=1}^{k} \neg link(X, E_i) \wedge \forall Z. \bigwedge_{i=1}^{k} \neg own(Z, E_i))$$

This implies that there exists an agent n s.t. $idle(n) \in \Delta$, and there exist $e_1, \ldots, e_n$ s.t.

$$link(m, e_i), own(m, e_i) \notin \Delta \; for \; i : 1, \ldots, k$$

and for any m occurring in Δ. The application of the rule yields a configuration Δ' defined as

$$\Delta' = \Delta \cup \{link(n, e_1), \ldots, link(n, e_k), own(n, e_1), \ldots, own(n, e_k), busy(n)\}$$

Since by assumption $own(m, e_i), link(m, e_i) \notin \Delta$ for any m occurring in Δ and $i : 1, \ldots, k$, we have that $\Delta' \models \Psi'$.

Now consider a configuration Δ with cardinality n and assume that $\Delta \models \Psi'$. Now consider s_k s.t. $k \leq n$. If s_k can be applied to Δ, there exists an interpretation σ s.t. $\Delta \models C_k\sigma$ where

$$C_k = (busy(X) \wedge \bigwedge_{i=1}^{k} link(X, E_i) \wedge own(X, E_i))$$

This implies that there exists an agent n s.t. $busy(n) \in \Delta$, and there exist $e_1, \ldots, e_n$ s.t.

$$link(n, e_i), own(n, e_i) \in \Delta \; for \; i : 1, \ldots, k$$

The application of the rule yields a configuration Δ' defined as

$$\Delta' = \Delta \setminus \{link(n, e_1), \ldots, link(n, e_k), own(n, e_1), \ldots, own(n, e_k)\} \cup \{idle(n)\}$$

Although the rule might remove a strict subset of $link$ and own predicates involving n, the resulting configuration still satisfies Ψ'.

Since Δ, k, and σ are chosen in arbitrary way, we have that the considered invariant Ψ' is an invariant for the whole family of rules of type s_k and r_k with $k \geq 0$.

Finally, we observe that for any Δ we have that if $\Delta \models \Psi'$ then $\Delta \models \Psi$. This proves that Ψ is still an invariant for DDP$^{\#}$. We conclude then by applying Prop. 5.1 obtaining correctness argument [1] be applied to DDP.

[1] that can

5.4. Related Work

We first focus our attention on a class of formal models for concurrent and distributed systems in which synchronization is achieved by using broadcast communication, a less standard communication primitive than rendez-vous or point-to-point communication. Rendez-vous communication involves a fixed a priori number of agents (e.g. a sender and a receiver in point-to-point communication). Properties of rendez-vous communication have been investigated deeply in the field of automated verification, see e.g. [17]. Interactions in models with broadcast communication are usually defined by allowing a finite but arbitrary number of agents to react to a given message or signal. This type of communication is particularly useful to define protocols for replicated systems, e.g. cache coherence protocols, algorithms with global conditions, e.g., simultaneous resets of local variables, and communication in an open environment like an Ad Hoc or Wireless network. Algorithmic verification of models with broadcast communication started receiving more attention after the introduction of the Broadcast Protocols of Emerson and Namjoshi [14]. FAST [1, 38], TRex [37], LASH [39], and MIST [31] are tools that can analyze counter abstractions of network models via encodings into vector addition systems and their extensions. For what concerns our case-study, a limit of these approaches is that they do not provide specification languages for topology-sensitive protocols. Indeed, the tools are specialized for handling models based on Petri Nets. MAP [15] is a tool based on transformations of constraint logic programs that can be applied to infinite-state systems with linear configurations and relations over data variables. MCMT [32] is a symbolic backward reachability engine based on SMT solvers that can handle parameterized systems with linear configurations. The MCMT tool is based on the EPR fragment of first order logic with arrays and applies different types of heuristics including invariant generation to reduce the state space. PFS [35] and UNDIP [36] are tool specifically devised to handle parameterized systems. MAPS, MCMT, PFS and UNDIP deal with systems with linear configurations, only. Concerning multithreaded programs, Boom [34] is a tool that applies symbolic algorithms, see, e.g., [21, 25, 26, 20], to verify counter abstractions of multithreaded programs. The algorithms behind the tool go beyond backward search. Indeed they combine several types of heuristics like those based on dynamic generation and refinement of over-approximations (defined in terms of upward closed set of states). PCW [33] is a tool that applies ordered counter abstraction [16], a refinement of monotonic abstraction with CEGAR, for the verification of parameterized systems. In this setting over-approximations are refined by using stronger and stronger orderings that can be used to define upward closed sets that "forbid" specific patterns (e.g. they forbid sets of points defined by a given equation). In view of the considered type of abstractions (Boolean progams, linear configurations) these tools do not seem adequate to model graph protocols.

AUGUR 2 [24] is a tool devised for the analysis of Graph Transformation Systems using approximated unfoldings based on Petri nets. The approximated systems can then be verified using regular expressions, first order logic and coverability checking techniques. AUGUR 2 does not handle global operations like those needed for modeling broadcast communication. PETRUCHIO [27] is a tool that extracts a Petri net representation from specifications of dynamic networks based on the π-calculus.

UNCOVER [30, 40] is a tool that performs a symbolic backward reachability analysis for GTS with universally quantified conditions. The tool exploits a generalization of monotonic abstraction to quantifications over graph patterns as a heuristic to manipulate infinite sets of configurations using minimal constraints (given in form of graphs) only. UNCONVER can be viewed as the counterpart of

UNDIP and PFS for systems in which configurations have a graph structure. Differently from [30, 40] and [28], our specification logic can be applied to define invariants involving an arbitrary but fixed number of nodes. For this purpose, we use assertions with parametric formulas. Global conditions can be defined using universally quantified formulas.

The use of SMT solvers in the style of the parallel verifier Cubicle [6] for graph-based specification is an interesting direction to explore.

There exist several proposals of formal languages for protocol specifications. GLog is inspired to languages based on multiset rewriting [5, 7], and evolving databases [19]. In [5] the authors adopted multiset rewriting on first order terms to specify security protocols. An extension of the language with constraints have been proposed in [7]. Undecidability results for restricted fragments (e.g. relabeling) can be reformulated for other restrictions of multiset rewriting rules like the balanced rules defined in [22, 23]. In [19] the authors introduce a logic-based language to specify evolving databases. In this setting conditions of update rules can be arbitrary queries on the current state (database). Undecidability results for problems like reachability have been studied for the above mentioned languages via reduction to Turing equivalent models (Turing machines, Minsky machines). Although the proof technique is similar in spirit, in this paper we do not study reachability problems. Indeed, we consider a decision problem in which (a) initial states are not fixed a priori (they are defined by an infinite set of configurations), and (b) consider a relaxed version of reachability problems that is very close to the coverability problem studied in Petri nets.

Concerning the considered case-study, for building a proof for the DDP case-study we took inspiration from methods applied in proof theory (e.g. to focus on special classes of rules) and concurrency (e.g. partial order reduction). The use of permutation schemes seems less standard in approaches based on deductive verification. In particular the other approaches applied to the formal verification of the DDP protocol [30, 40, 28] are based on much more intricate machineries (assume-guarantee methods with complex circular proof systems, or symbolic backward analysis based on graph transformation systems). The use of a specification logic based on simple conditional update rules, and of source to source transformations help us in providing a much simpler proof of the desired correctness properties.

6. Conclusions

We have presented a formal language for the specification of asynchronous distributed systems based on logic-based update rules. For the considered language, we have presented a verification approach that combines transformation schemes based on commutation properties and inductive verification. The proposed approach can be applied to verify the correctness of a distributed version of the dining philosopher protocol regardless the network topology and the number of nodes. Mechanization of the considered proof method could be an interesting direction for future work. One step could be that of introducing proof rules to reason symbolically on sets of configurations that represent graphs and on their structural properties (paths, link, cycles). In [9] the author introduces an assertional language with $path$-predicates defined for configurations of GLog specification. $path$-predicates can be used to define sets of configurations that satisfy reachability properties between nodes with certain labels. This kind of assertions can be used to reason about more complex structural properties of distributed

protocols (e.g. loop-freedom, existence of connection paths). Symbolic methods for automatically reason about this kind of properties have very high complexity due to a possible combinatorial explosion when matching them against symbolic representations of configurations. More investigations seem needed in order to find restricted assertional languages (e.g. using path constraints as in CLP languages) with a reasonable complexity.

References

[1] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.

[2] N. Bertrand, G. Delzanno, B. König, A. Sangnier, and J. Stückrath. On the decidability status of reachability and coverability in graph transformation systems. In *RTA'12*, volume 15 of *LIPIcs*, pages 101–116. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[3] N. Bertrand, G. Delzanno, B. König, A. Sangnier, and J. Stückrath. On the decidability status of reachability and coverability in graph transformation systems. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, pages 101–116, 2012.

[4] N. Bertrand, P. Fournier, and A. Sangnier. Distributed local strategies in broadcast networks. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, pages 44–57, 2015.

[5] I. Cervesato, N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop, CSFW 1999, Mordano, Italy, June 28-30, 1999*, pages 55–69, 1999.

[6] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi. Cubicle: A parallel smt-based model checker for parameterized systems - tool paper. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 718–724, 2012.

[7] G. Delzanno. Constraint-based automatic verification of abstract models of multithreaded programs. *TPLP*, 7(1-2):67–91, 2007.

[8] G. Delzanno. A logic-based approach to verify distributed protocols. In *Proceedings of the 31st Italian Conference on Computational Logic, Milano, Italy, June 20-22, 2016.*, pages 86–101, 2016.

[9] G. Delzanno. Reachability predicates for graph assertions. In *Reachability Problems - 10th International Workshop, RP 2016, Aalborg, Denmark, September 19-21, 2016, Proceedings*, pages 63–76, 2016.

[10] G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, pages 289–300, 2012.

[11] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, pages 313–327, 2010.

[12] G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 441–455, 2011.

[13] G. Delzanno, A. Sangnier, and G. Zavattaro. Verification of ad hoc networks with node and communication failures. In *FORTE/FMOODS'12*, volume 7273 of *LNCS*, pages 235–250. Springer, 2012.

[14] E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Thirteenth Annual IEEE Symposium on Logic in Computer Science, Indianapolis, Indiana, USA, June 21-24, 1998*, pages 70–80, 1998.

[15] F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Improving reachability analysis of infinite state systems by specialization. *Fundam. Inform.*, 119(3-4):281–300, 2012.

[16] P. Ganty and A. Rezine. Ordered counter-abstraction - refinable subword relations for parameterized verification. In *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, pages 396–408, 2014.

[17] S. M. German and A. P. Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

[18] S. Ghilardi and S. Ranise. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *Logical Methods in Computer Science*, 6(4), 2010.

[19] B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 163–174, 2013.

[20] A. Kaiser, D. Kroening, and T. Wahl. Lost in abstraction: Monotonicity in multi-threaded programs. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, pages 141–155, 2014.

[21] A. Kaiser, D. Kroening, and Thomas Wahl. A widening approach to multithreaded program verification. *ACM Trans. Program. Lang. Syst.*, 36(4):14, 2014.

[22] M. I. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Bounded memory dolev-yao adversaries in collaborative systems. In *Formal Aspects of Security and Trust - 7th International Workshop, FAST 2010, Pisa, Italy, September 16-17, 2010. Revised Selected Papers*, pages 18–33, 2010.

[23] M. I. Kanovich, P. D. Rowe, and A. Scedrov. Collaborative planning with confidentiality. *J. Autom. Reasoning*, 46(3-4):389–421, 2011.

[24] B. König and V. Kozioura. Augur 2 - A new version of a tool for the analysis of graph transformation systems. *Electr. Notes Theor. Comput. Sci.*, 211:201–210, 2008.

[25] D. Kroening. Automated verification of concurrent software. In *Reachability Problems - 7th International Workshop, RP 2013, Uppsala, Sweden, September 24-26, 2013 Proceedings*, pages 19–20, 2013.

[26] P. Liu and T. Wahl. Infinite-state backward exploration of boolean broadcast programs. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, pages 155–162, 2014.

[27] R. Meyer and T. Strazny. Petruchio: From dynamic networks to nets. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 175–179, 2010.

[28] K. S. Namjoshi and R. J. Trefler. Uncovering symmetries in irregular process networks. In *VMCAI*, pages 496–514, 2013.

[29] K. S. Namjoshi and R. J. Trefler. Analysis of dynamic process networks. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 164–178, 2015.

[30] J. Stückrath. Uncover: Using coverability analysis for verifying graph transformation systems. In *Graph Transformation - 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings*, pages 266–274, 2015.

[31] `https://github.com/pierreganty/mist`.

[32] `http://users.mat.unimi.it/users/ghilardi/mcmt/`.

[33] `http://www.ahmedrezine.com/tools/`.

[34] `http://www.ccs.neu.edu/home/wahl/Research/boom-and-cutoffs.html`.

[35] `http://www.it.uu.se/research/docs/fm/apv/tools/pfs/`.

[36] `http://www.it.uu.se/research/docs/fm/apv/tools/undip/`.

[37] `http://www.liafa.jussieu.fr/~sighirea/trex/`.

[38] `http://www.lsv.ens-cachan.fr/Software/fast/`.

[39] `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[40] `http://www.ti.inf.uni-due.de/de/research/tools/uncover/`.