Toward Automated Abstraction for Protocols on Branching Networks

Michael D. Jones
mike.jones@byu.edu

Ganesh Gopalakrishnan

Follow this and additional works at: https://scholarsarchive.byu.edu/facpub

Part of the Computer Sciences Commons

Original Publication Citation

BYU ScholarsArchive Citation
Jones, Michael D. and Gopalakrishnan, Ganesh, "Toward Automated Abstraction for Protocols on Branching Networks" (2000). All Faculty Publications. 587.
https://scholarsarchive.byu.edu/facpub/587

This Peer-Reviewed Article is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Faculty Publications by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Toward Automated Abstraction for Protocols on Branching Networks

Michael Jones, Ganesh Gopalakrishnan
School of Computing, University of Utah
mjones, ganesh@cs.utah.edu

Abstract — We have used various manual abstraction techniques to formally verify a transaction ordering property for an IO protocol over bus/bridge networks. In the context of network protocol verification, an abstraction is needed to reduce the unbounded number of network configurations to a small number of representative networks that can be checked using algorithmic methods. The manually derived abstraction was both brittle and difficult to validate. In this report, we discuss the need for abstraction techniques in the formal verification of protocols over networks and present our recent efforts to create an automatic abstraction technique for network protocols using predicate abstraction as a starting point.

Keywords — Formal verification methods, parameterized systems, predicate abstraction

We address the problem of abstraction in the formal verification of safety properties at the bus/bridge level for protocols defined over acyclic branching networks. Abstraction is needed in this context because formal methods applied directly to protocols over networks are either not applicable, or at best difficult and time-consuming. The main source of difficulty is the unbounded nature of branching networks. Because there are an unbounded number of configurations that must be checked, it is not possible to apply algorithmic methods, such as model checking, to all possible network configurations. We have found [MHJG00] that it is prohibitively difficult to apply interactive theorem proving in the context of protocols on branching networks.

In this report, we discuss our recent efforts to increase the amount of automation available for creating and reasoning about abstractions of protocols defined over branching networks. The problem with manually derived abstractions is that they need to be validated. Validating an abstraction is the process of showing which properties are preserved by the abstract model. We have found [JG00] that building a validation proof for a manually derived abstraction for protocols over networks is also difficult.

The significance of this work is that it will provide a technique for creating abstractions of protocols over branching networks such that certain properties of the protocol can be checked with minimal manual effort. The novel feature of this work is a predicate abstraction technique suitable for use on protocols which are defined over networks in which the states and connectivity of intermediate nodes affect the property being checked.

We begin by reviewing relevant results from predicate abstraction and parameterized system verification. Section II contains a detailed presentation of the abstraction scheme. The formal presentation in section II is closely patterned after the presentation in [GS97]. Section III gives an example, and we close with our thoughts on abstractions for networks in the final section.

I. RELATED WORK

The ideas presented here are an extension of predicate abstraction as described in [GS97]. Predicate abstraction is a form of abstraction in which the reduction is constructed manually but the validation and checking tasks are automated. To create a predicate abstraction, the user specifies a set of predicates which are used to divide the abstract state space depending on the truth, or falsity, of the predicates. Given the predicates, an abstract transition relation is constructed such that the original infinite state model is a refinement of the abstract finite state model. In practice, predicate abstraction requires several rounds of verification in which the user adds invariants to eliminate false negatives between rounds. The PVS theorem prover contains support for predicate abstraction and invariant strengthening [SS99]. Predicate abstraction has been applied to limited forms of networks [LS97] in which the particular shape of the network is unimportant to the property being verified. A form of predicate abstraction using BDDs to represent sets of reachable states has been implemented using the Murφ model checker [DDP99]. It is difficult to apply predicate abstraction to parameterized systems due to the quantification required to describe arrays of processes. Quantification makes predicate abstraction difficult because determining if a transition is enabled for a state description containing quantification is either undecidable or requires a complex decision procedure. We address the quantification problem by introducing a second variable for each predicate that encodes whether or not a node exists in the array that satisfies the predicate.

The proposed abstraction scheme applies to branching parameterized systems. While the verification problem for parameterized systems is in general undecidable [AK86], methods for reasoning about limited classes of protocols and properties have been developed. Other methods for reasoning about parameterized systems are based on regular expressions [KMM+97], [SG89], [CG95], rather than abstract state variables. While regular expression based techniques have found success primarily in linear or ring shaped topologies (see [BJNT00], [PS00] for recent results), the early papers consider extensions to include reasoning about branching topologies. Despite the ability of regular expression representations to describe and reason about branching topologies, no results for complex protocols have
yet appeared in the literature. By using a different representation and abstraction scheme over a limited class of properties, we hope to derive an abstraction scheme that can be applied to larger examples such as commercial multi-bus IO protocols. The added restriction in our work is that we consider only safety properties defined over a constant number of terminal nodes rather than safety properties in general.

II. Predicate Abstraction for Networks

The idea behind the abstraction scheme is to project the subnetwork containing the terminal nodes in the property being checked then model the state of each network segment using a variant of predicate abstraction. The variant of predicate abstraction uses an additional boolean variable for each predicate to indicate whether or not a node exists which satisfies the predicate represented by the state variable. The additional existence variable is used to encode quantification which is required to model parameterized systems using predicate abstraction. In the remainder of this section we formally define the model of computation and abstraction scheme. The next section contains an example of the abstraction scheme applied to a trivial property on a simplified version of the PCI protocol.

A network of processes is created by instantiating a parameterized finite state protocol \( p_i \) at each node. A routing table defines the connectivity in the network.

Definition 1: Network of nodes. A network of nodes is a six-tuple \((N, A, R, X, T, I)\) where:

- \( N = 1 \ldots n \), a list of node indices.
- \( A \subseteq N \), a list of agent, or terminal, nodes.
- \( R = \{(A_i, A_j, n_1, n_2, \ldots, n_l)\} \) for agents \( A_i, A_j \) in \( A \), a routing table in which agent \( A_i \) is connected to agent \( A_j \) by the path through nodes \( n_1 \) through \( n_l \).
- \( X = X_1, X_2, \ldots, X_n \), the network state formed by taking the initial state of each node, \( X_i \).
- \( T = T_1, T_2, \ldots, T_n \), the set of network transitions formed by the transitions, \( T_i \), from \( p_i \) at each node.
- \( I^j = I^1, I^2, \ldots, I^l \), the \( j \)th initial network state formed by taking the initial state of each node, \( I_j \).

The first three elements of a network, \( N, A, R \) define the global network structure. \( N \) and \( A \) define the interior and exterior nodes of the network while \( R \) defines the connections between exterior nodes using paths of interior nodes. The latter three elements, \( X, T, I^j \), define the state and transition relation of the network in a per-process fashion. Each of \( X, T, I^j \) are defined using the state, transitions and initial state of each node. The state, transitions and initial state of each node are all parameterized definitions, as given next.

Definition 2: Parameterized Protocol. A finite state parameterized protocol, \( p_i \) is a triple \((X_i, T_i, I_i)\) where:

- \( X_i = x_1, x_1 \ldots, x_n \), a list of declarations in which each variable \( x_i \) at node \( i \) has type \( t_j \).
- \( T_i = T^1_i, T^2_i, \ldots, T^m_i \), a list of transitions for process \( i \) in which each \( T^j_i \) has the form \( T^j_i = g_j(i, R, X) \rightarrow asgn(i, R, X) \).
- \( I_i^j = asgn X_i, \) the \( j \)th initial state for process \( i \) in which each variable in \( X_i \) is assigned a value.

Both the guard and the assignment in each transition take the routing table and network state as parameters. Including the network state and routing tables as parameters allows the guard and assignment to access the states of other nodes. For example, the guard in transition \( T^j \) can use the state of a node to determine if \( T^j \) is enabled. Similarly, the presence of the routing table and network work state in the update expression allows the transition at a node to modify the states of other nodes in the network. Transitions need to modify the state of adjacent nodes if the transition moves a message between two nodes.

We now define the semantics of a network of processes over a state graph.

Definition 3: Parameterized State Graph. Given a network of processes, and a routing table \( R \), the parameterized state graph, \( S^R \) is a three-tuple, \((Q^R, P^R, I^R)\) where

- \( Q^R = x_1, x_1 \ldots, x_n, x_1^2, x_2^2, \ldots, x_l^2, x_1^3, x_2^3, \ldots, x_l^3 \) ~
- \( P^R = \cup_{j=1}^{l} I^j(q) \) where

\[
\tau^P_j(q) = \begin{cases} 
q & \text{if } g^j(q, R, X) = F \\
\text{asgn}^j(q, R, X) & \text{otherwise}
\end{cases}
\]

\( I^R = \{qinit(q)\} \)

In the definition of \( S^R \), \( Q^R \) is a graph node representing a global network state, \( P^R \) represents the transitions enabled for a graph node and \( I^R \) is the set of initial states. The entire state graph, \( S \), for a protocol defined over a network of processes is created by taking the union of all state graphs over all routing tables. For networks with unboundedly large network topologies, \( S \) contains a finite but unbounded number of states. Since the unboundedness of \( S \) stems from the unbounded number of unique valid routing tables, the crux of the abstraction is creating a model which contains a bounded number of network topologies and using predicate abstraction to model the unbounded number of states in each topology. Each topology has an unbounded number of states because there may be an unbounded number of nodes in each edge of a given network topology.

We now define the abstraction of a state graph given the abstraction of a routing table. This part of the abstraction uses predicate abstraction to represent the unbounded number of states in a topology. The abstraction of routing tables will be discussed later. We begin with preliminary definitions and results for predicate transformers.

Definition 4: Strongest Postcondition. For a parameterized transition \( \tau^n_i \) and a set of states characterized by a predicate \( \varphi \), the strongest postcondition of \( \varphi \) by \( \tau^n_i \) is defined as:

\[
\text{post}[\tau^n_i](\varphi) = \exists q', \tau^n_i(q', q) \wedge \varphi(q')
\]

Definition 5: Weakest Precondition. For a parameterized transition \( \tau^n_i \) and a set of states characterized by a predicate \( \varphi \), the weakest precondition of \( \varphi \) by \( \tau^n_i \) is defined as:

\[
\text{pre}[\tau^n_i](\varphi) = \forall q', \tau^n_i(q', q) \Rightarrow \varphi(q')
\]

Intuitively, \( \text{post}[\tau^n_i](\varphi) \) is the smallest set of successors of \( \varphi \) reachable by \( \tau^n_i \) and \( \text{pre}[\tau^n_i](\varphi) \) is the largest set of states
that map to \( \varphi \) by \( \tau_n^m \). As reported in [GS97], we can rewrite \( \overline{\text{pred}}[\tau_n^m](\varphi) \) and \( \text{post}[\tau_n^m](\varphi) \) as

\[
\overline{\text{pred}}[\tau_n^m](\varphi) = (y^*[\varphi, n, R, X] / (n, R, X))
\]

\[
\text{post}[\tau_n^m](\varphi) = \varphi'[\varphi, \overline{\text{pred}}[\tau_n^m](\varphi')]
\]

**Definition 6:** Abstract State Graph. Given a state graph \( S^R = (Q^R, U^R, R') \) for a network of nodes and routing table \( R \) with abstracted routing table \( R' \) (as will be defined later), let \( Q_A^R \) be a lattice of abstract states with \( S_A^R = (Q_A^R, U_A^R, R') \) and \( (\alpha : 2^R \rightarrow Q_A^R, \gamma Q_A^R \rightarrow 2^Q) \). Then \( S_A^R \) is an abstraction of \( S^R \) iff

- \( I^R \subseteq \gamma(I_R^R) \), and
- \( \forall n, \forall q_A \in Q_A^R \)

\[
\text{post}[\tau_n^m]([\gamma(q_A)]) \subseteq \gamma([\text{path}(n)])
\]

\[
R \subseteq \gamma(R')
\]

Intuitively, \( S_A^R \) is an abstraction of \( S^R \) if for every transition at any node in \( S^R \), the strongest postcondition of the abstract state in \( \gamma(q_A) \) is a subset of the concrete states in \( \gamma([\text{path}(n)]) \). In short, for every move \( \tau \) at any node in \( S^R \), there is a move \( \tilde{\tau} \) in \( S_A^R \) that does as much or more than \( \tau \).

Next we identify an abstraction scheme for \( S^R \). We consider the abstract state lattice \( Q_A^R \) induced by the predicates \( \varphi, \varphi^2, \ldots, \varphi^m \) parameterized by each node \( n \). For each predicate in \( \varphi \) and abstract path \( j \) in \( R \) we define two abstract state variables \( \tilde{\varphi}_j^m \) and \( \tilde{\varphi}_j^m \).

**Definition 7:** Abstract state lattice. Given the set of monomials over \( \varphi \) and \( \varphi^2 \) for an abstract routing table \( R' \) and glb operator \( \wedge \) (boolean conjunction) with lub operator \( \vee \) (weakened form of disjunction), the abstract state lattice \( M^R \) is defined to be \( (\tilde{\varphi}_j^m \wedge \varphi_j^m) \).

Each monomial in \( M^R \) represents a set of concrete states described by a conjunction of quantified predicates as given in Figure 1. As shown in the figure, an abstract monomial represents the set of concrete states which contain sequences of nodes that may contain a node that satisfies a predicates \( \varphi \) depending on the values of \( \tilde{\varphi}_j^m \) and \( \varphi_j^m \) in the monomial.

We now define the abstract transition relations parameterized by paths using monomials over abstract state variables. For a given abstract state \( \tilde{\varphi}_j^m \) we compute the abstract transition \( \tilde{\tau}_j^m \) corresponding to transition \( \tau_n^m \) for some \( n \in k \) using the equation shown in Figure 2. The equation states that the next state of \( \tilde{\tau}_j^m \) is in \( F \) if \( \gamma(n) \) always fails to satisfy the guard for transition \( \tau_n^m \) for any \( n \in k \); but that \( \tilde{\tau}_j^m \) includes any disjunction over \( \tilde{\varphi}_j^m \), \( \tilde{\varphi}_j^m \), \( \tilde{\varphi}_j^m \), and \( \tilde{\varphi}_j^m \) for next states that may or may not have nodes in path \( k \) that may or may not satisfy \( \varphi_j^m \). Note that each \( \gamma(n) \wedge \varphi_j^m \Rightarrow \neg \varphi_j^m \) is simply the quantifier free form of \( \gamma(n) \Rightarrow \varphi_j^m \).

As mentioned before, the crux of the reduction is representing the potentially unbounded number of routing tables over an unbounded set of terminal and non-terminal nodes to a bounded number of small abstract routing tables that represent all possible network configurations. If we restrict the set of invariants to safety properties over a constant number of terminal nodes, then we can enumerate all possible (up to path length) acyclic configurations, \( R_n \), on \( n \) terminal nodes. The class \( R_n \) of networks corresponds to the class of unique Steiner topologies over \( n \) terminals where each terminal has degree one (see [HRW92] for a discussion of Steiner topologies). This correspondence gives an upper-bound on the number of unique acyclic network configurations over \( n \) nodes.

Given a concrete routing table \( R \), we construct a corresponding abstract routing table \( R' \) by first labeling \( n \) of the terminal nodes (as required to check a property on \( n \) terminal nodes). The abstract routing table \( R' \) is then created by projecting only the \( n \) labeled nodes and the paths between them. The final step is to replace the nodes in each unique path segment between the \( n \) labeled nodes with a single path variable. An example of creating an abstract routing table can be found in Section III. An important property of the routing table abstraction is that all networks with the same topology on the \( n \) labeled nodes are reduced to the same abstract routing table.

Given an abstract routing table \( R' \in R_n \) we construct the concretization of \( R' \), \( \gamma(R') \), as follows. The set \( R_n \) is the set of concrete routing tables such that:

- Every terminal node \( t_i \) in \( R' \) also appears in every \( R_n \).
- If \( R'(i,j) = p_1 \ldots p_m \), then \( R(i,j) = \gamma(p_1) \ldots \gamma(p_m) \) where \( \gamma(p) \) is the smallest set of nodes that appears in every entry of \( R \) which corresponds to an entry in \( R' \) containing \( p \).

More formally, we define \( \gamma(R') \) as

**Definition 8:** Concretization of a Routing Table. The concretization of a routing table \( R' \) is

\[
\{ R \mid t_i \in R' \Rightarrow t_i \in R \wedge R'(i,j) = p_1 \ldots p_m \Rightarrow R(i,j) = \gamma(p_1) \ldots \gamma(p_m) \}
\]

where \( \gamma(p) = \{n_1 \ldots n_m \mid k, l, p \in \gamma(k,l) \Rightarrow n_1 \ldots n_m \in R(k,l)\} \)

**III. Tutorial Example**

We give a simple example of checking a property on a simplified version of the PCI protocol [PCI95], called PCI_p, including only posted messages. In PCI, a posted message is an unacknowledged message which can be neither deleted nor reordered. The property we wish to check is: “if two agents send a posted message to a single destination, then one of the messages always reaches the destination first.” The property is not true for PCI_p so we expect to find a violation.

There is only one topologically unique way to connect three nodes in an acyclic network, so there is only one abstract routing table to consider in the construction of the abstract state graph. Although all networks over three terminal nodes have the same abstraction, we provide an example of constructing the abstract routing table. Suppose we have a PCI_p network containing three terminal nodes with the following routing table:
We chose to mark nodes 1, 2, and 3 as the terminal nodes in the property being checked; that means that node 10 will be eliminated in the abstraction. After eliminating terminal node 10, we replace the nodes in each non-branching sub-path in the remaining network with a single path. For example, all paths to and from node 1 include nodes 1 and 2. This means that nodes 1 and 2 form a non-branching sub-path, which we represent as path $A$ in the abstract routing table. Note that the abstract routing table $R^A$ is defined such that $R_{ij}^A = \begin{cases} 1 & \text{if } i < j, \\ \infty & \text{otherwise} \end{cases}$.

Fig. 1. Sets of concrete states represented by a monomial over the abstract state variables.

FIG. 2. Computing the abstract transition relation for a concrete transition $T_i$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4,5,6,7</th>
<th>4,5,8,9</th>
<th>4,5,12,11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>2,6,5,4</td>
<td>-</td>
<td>4,5,6,7</td>
<td>4,5,12,11</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7,6,5,4</td>
<td>-</td>
<td>4,5,6,7</td>
<td>4,5,12,11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9,8,5,4</td>
<td>4,5,6,7</td>
<td>-</td>
<td>4,5,12,11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11,12,5,4</td>
<td>11,12,6,7</td>
<td>11,12,8,9</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We have shown that the concretization of the abstract initial state does not satisfy $\neg g_n^A$ for every node $n$ in $\gamma(A)$. We check the negated form of the previous equation:

$$\gamma(A) \Rightarrow \exists n \in A. \neg g_n^A$$

which is trivially true. This gives us that $\tilde{T}_1^A \neq F$ and we compute the values of the state variables for the other predicates and network paths. We compute the effects of $\tilde{T}_1^A$ by considering the effects of applying $\tilde{T}_1^A$ to any node $n$ in the concretization of $A$. We present in detail how the values of $\tilde{d}_1^A, \tilde{q}_1^A$ are computed and summarize the values.
for the other state variables.

First, we check if $\delta_A^1\phi_A^1$ should be added to the next state of $\hat{\tau}_1^A(\hat{I}_1)$. We add $\delta_A^1\phi_A^1$ if

$$\exists m \in A. \exists n \in A. (\gamma(\hat{I}_1) \wedge g^n) \Rightarrow \phi_1^m[\text{asn}_{g^n}(x)/x]$$

It is possible to pick a node $n$ in $\gamma(A)$ such that $\phi_1^m$. Suppose $n$ is not the last node in $\gamma(A)$ and $m$ is the next node after $n$ in $\gamma(A)$, then $\text{asn}_{g^n}(x)$ moves $P_1$ from node $n$ to node $m$. In this case, if node $m$ is empty before appending $P_1$ then $\phi_1^m$ is true of $m$ after $\text{asn}_{g^n}(x)/x$.

Next, we check if $\delta_A^1\phi_A^1$ should be added to the next state of $\hat{\tau}_1^A(\hat{I}_1)$. We add $\delta_A^1\phi_A^1$ if

$$\exists n \in A. \exists m \in A. (\gamma(\hat{I}_1) \wedge g^n) \Rightarrow -\phi_1^m[\text{asn}_{g^n}(x)/x]$$

Which is also true. If we pick $n$ and $m$ to be the same node in $\gamma(A)$, then after $\text{asn}_{g^n}(x)/x) P_1$ is no longer in $m$ so $\exists m \in A. -\phi_1^m[\text{asn}_{g^n}(x)/x]$ and we add $\delta_A^1\phi_A^1$ to $\hat{\tau}_1^A(\hat{I}_1)$.

Now we check if $\delta_A^1\phi_A^1$ should be added to the next state of $\hat{\tau}_1^A(\hat{I}_1)$. We add $\delta_A^1\phi_A^1$ if

$$\exists n \in A. \exists m \in A. (\gamma(\hat{I}_1) \wedge g^n) \Rightarrow -\phi_1^m[\text{asn}_{g^n}(x)/x]$$

Which is true if we pick $n$ to be the last node in $\gamma(A)$. In this case, $\text{asn}_{g^n}(x)/x)$ moves $P_1$ out of the last node in $\gamma(A)$ so that $\exists m \in A. -\phi_1^m[\text{asn}_{g^n}(x)/x]$. So we add $\delta_A^1\phi_A^1$ to $\hat{\tau}_1^A(\hat{I}_1)$.

Finally, we check if $\delta_A^1\phi_A^1$ should be added to the next state of $\hat{\tau}_1^A(\hat{I}_1)$. We add $\delta_A^1\phi_A^1$ if

$$\exists n \in A. \exists m \in A. (\gamma(\hat{I}_1) \wedge g^n) \Rightarrow -\phi_1^m[\text{asn}_{g^n}(x)/x]$$

Which is not true because $P_1$ is in at most one node in $\gamma(A)$ before and after $\text{asn}_{g^n}(x)/x)$ and $P_1$ is in different nodes before and after the transition. Consequently, $P_1$ can not be in all nodes of $\gamma(A)$ after the transition.

We now have that $\delta_A^1\phi_A^1$, $\delta_A^1\phi_A^1$, and $\delta_A^1\phi_A^1$ are in $\hat{\tau}_1^A(\hat{I}_1)$. We have now computed the values of $\delta_1$ and $\phi_1$ for path $A$. We next perform a similar analysis for paths $B$ and $C$.

- For path $B$, only $\delta_B^1\phi_B^1$ is in $\hat{\tau}_1^B(\hat{I}_1)$. This is because $\text{asn}_{g^n}(x)/x)$ can not move message $(1,3)$ to a bridge in path $B$ since next $(i,3)$ is never a bridge in $B$ for all bridges $i$ in path $A$.

- For path $C$, we find that the following terms for $\tau_1^C$ are in $\hat{\tau}_1^C(\hat{I}_1)$: $\delta_C^2\phi_C^0$, $\delta_C^2\phi_C^0$ and $\delta_C^2\phi_C^0$. These terms are included because $\text{asn}_{g^n}(x)/x)$ can, but is not required to, move message $(1,3)$ to a bridge in path $C$ from a bridge in path $A$.

Next, we need to determine which state variables for predicates $\phi_1$ and $\phi_2$ for bridges in paths $A$, $B$, and $C$ need to be included in $\hat{\tau}_1^A(\hat{I}_1)$.

- For predicate $\phi_0$, $\hat{\tau}_1^A(\hat{I}_1)$ has no effect on $\phi_0$ because $\text{asn}_{g^n}(x)/x)$ can not violate the invariant. Suppose $\text{asn}_{g^n}(x)/x)$ moves $(1,3)$ into the head of agent 3. Then if $\phi_0$ was true before $\text{asn}_{g^n}(x)/x)$ then $\phi_0$ is still true after $\text{asn}_{g^n}(x)/x)$.

For predicate $\phi_2$, $\hat{\tau}_1^A(\hat{I}_1)$ has no effect on $\phi_2$ either. This is because $\text{asn}_{g^n}(x)/x)$ does not affect message $(2,3)$. While $\text{asn}_{g^n}(x)/x)$ moves message $(1,3)$, the location of message $(2,3)$ has no bearing on the value of $\phi_2$.

This concludes the computation of $\hat{\tau}_1^A(\hat{I}_1)$. We have that $\hat{\tau}_1^A(\hat{I}_1)$ includes the following boolean expression over the abstract state variables:

$$\hat{\tau}_1^A(\hat{I}_1) = \neg\delta_A^1\phi_A^1 \wedge \neg\delta_B^1\phi_B^1 \wedge \neg\delta_C^1\phi_C^0 \wedge (\delta_A^1\phi_A^1 \vee \delta_B^1\phi_B^1 \vee -\delta_C^1\phi_C^0) \land \neg\delta_A^1\phi_A^1 \wedge (\delta_B^1\phi_B^0 \vee \delta_B^1\phi_B^0 \vee -\delta_C^1\phi_C^0) \land \neg\delta_A^1\phi_A^0 \wedge \delta_B^1\phi_B^2 \wedge \delta_B^1\phi_B^2 \wedge -\delta_C^1\phi_C^0$$

Next, we perform the same analysis for $\hat{\tau}_1^B(\hat{I}_1)$ and $\hat{\tau}_1^C(\hat{I}_1)$. We begin by checking that

$$\gamma(\hat{I}_1) \Rightarrow \forall n \in B. -\phi_2^n \wedge \gamma(\hat{I}_1) \Rightarrow \forall n \in C. -\phi_2^n$$

Both of these implications are true because $-\delta_B^1\phi_B^0$ and $-\delta_C^1\phi_C^0$ are in $\hat{I}_1$. For $\hat{\tau}_1^B(\hat{I}_1)$, we then have that

$$\exists m \in B. \phi_2^m \land \gamma(\hat{I}_1) \Rightarrow \forall n \in C. -\phi_2^n$$

and $\phi_2^m = \phi_2^n$. Because no nodes in paths $B$ and $C$ satisfy the guard on $\tau_1$ in state $I_1$, both $\hat{\tau}_1^B(\hat{I}_1)$ and $\hat{\tau}_1^C(\hat{I}_1)$ equal $F$.

Having determined the abstraction of all possible effects of $\tau_1$ on all network nodes of all concrete states represented by $I_1$, we now analyze the effects of $\tau_2$ on all nodes in all states represented by $I_1$. The effects of $\tau_2$ are similar to the effects of $\tau_1$ except $\tau_2$ moves message $(2,3)$ rather than message $(1,3)$. We will compute the next state set for $\hat{\tau}_2^B(\hat{I}_1)$ and find a violation of the invariant. To make the example complete, we first check that the transition in enabled by attempting to disprove

$$\gamma(\hat{I}_2) \Rightarrow \forall n \in B. -\phi_2^n$$

Since $\delta_B^1\phi_B^0$ is in $\hat{I}_1$, we have that

$$\exists m \in B. \phi_2^m \land \gamma(\hat{I}_2) \Rightarrow \forall n \in B. -\phi_2^n$$

which is false since $\phi_2 = \phi_2^m$. Since $\hat{\tau}_2^B(\hat{I}_1)$ is enabled, we next compute which abstract state variables are in the next state for $\hat{\tau}_2^B(\hat{I}_1)$.

We will start with $\delta_C^0\phi_C^0$ because it will yield a violation of the form $\phi_2^0$. To include $\delta_C^0\phi_C^0$ in the next state of $\hat{\tau}_2^B(\hat{I}_1)$ we need to prove that

$$\exists n \in B. \exists m \in C. \gamma(\hat{I}_1) \wedge g^n \Rightarrow -\phi_2^m[\text{asn}_{g^n}(x)/x]$$

If we pick $n$ to be the last bridge in $B$ and let $m$ be bridge 3 in path $C$ (implying that $C$ has only one bridge, which is an acceptable assumption) then $\text{asn}_{g^n}(x)/x)$ moves message $(2,3)$ into the head of the queue in $m$. Bridge $m$ now violates

$$x^2.q[0] = (2,3) \land (i = 3) \Rightarrow x^2.q[0] = ((1,3) \lor \text{empty})$$

so we add $\delta_C^0\phi_C^0$ to $\hat{\tau}_2^B(\hat{I}_1)$ and find that a node exists in path $C$ which violates $\phi_0$.
Predicate abstraction is conservative, rather than exact, so this violation does not necessarily imply that a violation exists in the concrete model. However, in this case, the violation can be translated into a violating trace. The violating PCI$_s$ trace allows the message (2,3) to reach agent 3 before the message (1,3).

IV. CONCLUDING REMARKS

By restricting the class of properties that can be checked and using a representation based on predicate abstraction, we have created an abstraction method which is intended to extend parameterized system verification to complex protocols over branching networks. At present, our predicate abstraction technique for networks exists only as “paperware.” That is, predicate abstraction for networks has been worked out on paper and pencil, and paper examples have been completed; but that we have not yet built a tool which incorporates these techniques and applies them systematically.

Before building a tool based on these ideas, we plan to extend the paper-ware version to include other network classes, apply to distributed shared memory systems and provide conservative support for reasoning about $\forall$CTL* properties. We anticipate that the most difficult aspect of the implementation will be providing reasoning support for deciding if the set of concrete states represented by a monomial of abstract state variables satisfies a guard to a transition.

REFERENCES


