2000-11-01

Toward Automated Abstraction for Protocols on Branching Networks

Michael D. Jones
mike.jones@byu.edu

Ganesh Gopalakrishnan

Original Publication Citation

M. Jones and G. Gopalakrishnan, "Toward Automated Abstraction for Protocols Over Branching Networks," IEEE International High Level Design Validation and Test Workshop (HLDVT'), pp. 147-154. Palo Alto, California, November 2.

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, tt 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], [CGJ95], 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

0-7695-0786-7/00 $10.00 © 2000 IEEE

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 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^j) where:

- $N = 1 \ldots n$, a list of node indices.
- $A \subseteq N$, a list of agent, or terminal, nodes.
- $R = \{(A_i, A_j, n_1, n_2 \ldots n_i) |$ 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_i.
- $X = X_1, X_2 \ldots X_n$, the network state formed by taking the 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.
- $I^j = I_1^j, I_2^j \ldots I_n^j$, the jth initial network state formed by taking the initial state of each node, I_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

- $X_i = x_1^i : t_1, x_2^i : t_2 \ldots x_n^i : t_n^i$, a list of declarations in which each variable x_j^i at node i has type t_j.
- $T_i = \tau_1^i, \tau_2^i \ldots \tau_m^i$, a list of transitions for process i in which each τ_j^i has the form $\tau_j^i = g_j(i, R, X) \mapsto asgn(i, R, X)$.
- $I_i^j = asgnX_i$, the jth 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 τ_j^i can use the state of an adjacent node to determine if τ_j^i 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

- $Q^R = x_1^1, x_2^1 \ldots x_n^1, x_1^2, x_2^2 \ldots x_n^2 \ldots x_1^i, x_2^i \ldots x_n^i$
- $P^R = \cup_{p=1}^{i}(\cup_{j=1}^{n} \tau_j^p(q)$ where
$$\tau_j^p(q) = \begin{cases} \perp & \text{if } g_j^p(q, R, X) = F \\ asgn_j^p(q, R, X) & \text{otherwise} \end{cases}$$
- $I^R = \{q | init(q)\}$

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 τ_i^n and a set of states characterized by a predicate φ, the strongest postcondition of φ by τ_i^n is defined as:

$$post[\tau_i^n](\varphi) = \exists q'.\tau_i^n(q', q) \wedge \varphi(q')$$

Definition 5: Weakest Precondition. For a parameterized transition τ_i^n and a set of states characterized by a predicate φ, the weakest precondition of φ by τ_i^n is defined as:

$$\widetilde{pre}[\tau_i^n](\varphi) = \forall q'.\tau_i^n(q, q') \Rightarrow \varphi(q')$$

Intuitively, $post[\tau_i^n](\varphi)$ is the smallest set of successors of φ reachable by τ_i^n and $\widetilde{pre}[\tau_i^n](\varphi)$ is the largest set of states

that map to φ by τ_i^n. As reported in [GS97], we can rewrite $\widetilde{pre}[\tau_i^n](\varphi)$ and $post[\tau_i^n](\varphi)$ as

$$\widetilde{pre}[\tau_i^n](\varphi) = (g_i^n(\bar{x}) \Rightarrow \varphi_i^n[asgn_i^n(n,R,X)/(n,R,X)])$$

$$post[\tau_i^n](\varphi) \Rightarrow \varphi' \text{iff } \varphi \Rightarrow \widetilde{pre}[\tau_i^n](\varphi')$$

Definition 6: Abstract State Graph. Given a state graph $S^R = (Q^R, \bigcup_{\tau_i^j}^R, I^R)$ 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 $S_A^{R'} = (Q_A^{R'}, \bigcup_{\hat{\tau}_i^j}^{R'}, I^{R'})$ and $(\alpha : 2^{Q^R} \mapsto Q_A^{R'}, \gamma Q_A^{R'} \mapsto 2^{Q^R})$. Then $S_A^{R'}$ is an abstraction of S^R iff

- $I^R \subseteq \gamma(I_A^{R'})$, and
- $\forall n. \forall i. \forall q_A \in Q_A^{R'}$.

$$post[\tau_i^n](\gamma(q_A)) \subseteq \gamma(\hat{\tau}_i^{\text{path(n)}}(q_A))$$

- $R \subseteq \gamma(R')$

Intuitively, $S_A^{R'}$ is an abstraction of S^R if: for every transition at every node in S^R, the strongest postcondition of the concrete states in $\gamma(q_A)$ is a subset of the concrete states in $\gamma(\hat{\tau}_i^{path(n)}(q_A))$. In short, for every move τ at every node in S^R, there is a move $\hat{\tau}$ in $S_R^{R'}$ that does as much or more than τ.

Next we identify an abstraction scheme for S^R. We consider the abstract state lattice $Q_A^{R'}$ induced by the predicates $\varphi_1^n, \varphi_2^n, \ldots \varphi_i^n$ parameterized by each node n. For each predicate in φ_i and abstract path j in R' we define two abstract state variables $\hat{\vartheta}_i^j \hat{\varphi}_i^j$.

Definition 7: Abstract state lattice. Given the set of monomials over $\hat{\vartheta}_i^j$ and $\hat{\varphi}_i^j$ for an abstract routing table R' and glb operator $\wedge$ (boolean conjunction) with lub operator $\sqcup$ (weakened form of disjuction), the abstract state lattice $\hat{M}^{R'}$ is defined to be $(\hat{m}, \wedge, \sqcup)$

Each monomial in $\hat{M}^{R'}$ 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 φ_n depending on the values of $\hat{\varphi}_n^k$ and $\hat{\vartheta}_n^k$ in the monomial.

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

As mentioned before, the crux of the abstraction is reducing 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 $R_\gamma = \gamma(R')$ is the set of concrete routing tables such that:

- Every terminal node t_i in R' also appears in every R_γ.
- 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 $\gamma(p) = \{n_1 \ldots n_m | \forall k,l.p \in R'(k,l) \Rightarrow n_1 \ldots n_m \in R(k,l)\}$

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:

$$\gamma(\hat{m}) = \forall k, n. \bigwedge_{\hat{\varphi}_n^k \in \hat{m}} \begin{cases} \text{if } \hat{m} \Rightarrow \hat{\vartheta}_n^k & \begin{cases} \exists i \in k. \neg\varphi_n^i & \text{if } \neg\hat{\varphi}_n^k \in \hat{m} \\ \exists i \in k. \varphi_n^i & \text{otherwise} \end{cases} \\ \text{if } \hat{m} \Rightarrow \neg\hat{\vartheta}_n^k & \begin{cases} \neg\exists i \in k. \neg\varphi_n^i & \text{if } \neg\hat{\varphi}_n^k \in \hat{m} \\ \neg\exists i \in k. \varphi_n^i & \text{otherwise} \end{cases} \end{cases}$$

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

$$\hat{\tau}_i^k(\hat{m}) = \begin{cases} F & \text{if } \gamma(\hat{m}) \Rightarrow \forall n \in k. \neg g_i^n \\ \bigwedge_{j=1}^{tq} \bigwedge_{l=1}^{p} \hat{m}_j^l & = \begin{cases} \hat{\vartheta}_j^l \hat{\varphi}_j^l & \text{if } \exists n \in k. \exists m \in l. (\gamma(\hat{m}) \wedge g_i^n) \Rightarrow \varphi_j^m[asgn_i^n(\bar{x})/\bar{x}] \\ \hat{\vartheta}_j^l \neg\hat{\varphi}_j^l & \text{if } \exists n \in k. \exists m \in l. (\gamma(\hat{m}) \wedge g_i^n) \Rightarrow \neg\varphi_j^m[asgn_i^n(\bar{x})/\bar{x}] \\ \neg\hat{\vartheta}_j^l \hat{\varphi}_j^l & \text{if } \exists n \in k. \neg\exists m \in l. (\gamma(\hat{m}) \wedge g_i^n) \Rightarrow \varphi_j^m[asgn_i^n(\bar{x})/\bar{x}] \\ \neg\hat{\vartheta}_j^l \neg\hat{\varphi}_j^l & \text{if } \exists n \in k. \neg\exists m \in l. (\gamma(\hat{m}) \wedge g_i^n) \Rightarrow \neg\varphi_j^m[asgn_i^n(\bar{x})/\bar{x}] \\ T & \text{otherwise} \end{cases} \end{cases}$$

Fig. 2. Computing the abstract transition relation for a concrete transition τ_i^n

	1	2	3	10
1	-	4,5,6,7	4,5,8,9	4,5,12,11
2	7,6,5,4	-	7,6,8,9	7,6,12,11
3	9,8,5,4	9,8,6,7	-	9,8,12,11
10	11,12,5,4	11,12,6,7	11,12,8,9	-

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 4 and 5. This means that nodes 4 and 5 form a non-branching sub-path, which we represent as path A in the abstract routing table. Note that the abstract routing table R' is defined such that $R \subseteq \gamma(R')$:

	1	2	3
1	-	A,B	A,C
2	B,A	-	B,C
3	C,A	C,B	-

A PCI_p message is written as a tuple (s, d) which indicates a message from agent s to agent d. We begin with the definition of the state and transitions at a PCI_p node, parameterized by node address i.

- X_i = q:Array[2] of PCI_p message, opposite_q: Array[2] of PCI_p message.
- Transition τ_1^i with
  $g_1^i = X_i.q[0] = (1,3) \wedge \text{next}(i,3) \neq 3 \wedge \neg\text{full}(X_{\text{next}(i,3)}.q)$
  $asgn_1^i = \text{delete}(0,i); \text{append}((1,3), \text{next}(i,3))$
- Transition τ_2^i with
  $g_2^i = X_i.q[0] = (2,3) \wedge \text{next}(i,3) \neq 3 \wedge \neg\text{full}(X_{\text{next}(i,3)}.q)$
  $asgn_2^i = \text{delete}(0,i); \text{append}((1,3), \text{next}(i,3))$
- I_1^i with $x^1.q[0] = (1,3)$, $x^2.q[0] = (2,3)$ and $x^j.q[k] = empty$ for every other j, k.

Each node contains two arrays which each contain a maximum of two messages. Transition τ_1^i moves message (1,3) from node i to the next node on the path from i to agent 3—if the queue in the next node is not full. Similarly, transition τ_2^2 moves message (2,3) to the next node on the path from i to agent 3.

The property we want to check asserts that the message from agent 1 to agent 3 arrives at agent 3 first. This is property is expressed by the following predicate:

$$\varphi_0^i = (i = 3) \Rightarrow x^i.q[0] = ((1,3) \vee empty)$$

We use φ_0^i and the additional predicates

$$\varphi_1^i = g_1^i = X_i.q[0] = (1,3) \wedge \text{next}(i,3) \neq 3 \wedge \neg\text{full}(X_{\text{next}(i,3)}$$
$$\varphi_2^i = g_2^i = X_i.q[0] = (2,3) \wedge \text{next}(i,3) \neq 3 \wedge \neg\text{full}(X_{\text{next}(i,3)}$$

to define the abstract state space. Given the initial state I_1 and the predicates $\varphi_0^i, \varphi_1^i, \varphi_2^i$ we have the following initial abstract state:

$$\hat{I}_1 = \begin{array}{l} \neg\hat{\vartheta}_0^A \neg\hat{\varphi}_0^A \ \neg\hat{\vartheta}_0^B \neg\hat{\varphi}_0^B \ \neg\hat{\vartheta}_0^C \neg\hat{\varphi}_0^C \\ \hat{\vartheta}_1^A \hat{\varphi}_1^A \ \neg\hat{\vartheta}_1^B \hat{\varphi}_1^B \ \neg\hat{\vartheta}_1^C \hat{\varphi}_1^C \\ \neg\hat{\vartheta}_2^A \hat{\varphi}_2^A \ \hat{\vartheta}_2^B \hat{\varphi}_2^B \ \neg\hat{\vartheta}_2^C \hat{\varphi}_2^C \end{array}$$

The initial abstract state represents all concrete states which contain message (1,3) in path A and message (2,3) in path B.

We compute the next states for $\hat{I}_1$ under the transitions τ_1^i and τ_2^i. We begin with $\hat{\tau}_1^A$. First, we check

$$\gamma(\hat{I}_1) \Rightarrow \forall n \in A. \neg g_1^n$$

that the concretization of the abstract initial state does not satisfy g_1^n for every node n in $\gamma(A)$. We check the negated form of the previous equation:

$$\gamma(\hat{I}_1) \Rightarrow \exists n \in A.\ g_1^n$$

Since $\hat{I}_1$ includes $\hat{\vartheta}_1^A \hat{\varphi}_1^A$ we have that $\exists n \in A. \varphi_1^n$ and from the definition of φ_1^n we have

$$\exists n \in A. g_1^n \Rightarrow \exists n \in A. g_1^n$$

which is trivially true. This gives us that $\hat{\tau}_1^A \neq F$ and we compute the values of the state variables for the other predicates and network paths. We compute the effects of $\hat{\tau}_1^A$ by considering the effects of applying τ_1^n to any node n in the concretization of A. We present in detail how the values of $\hat{\vartheta}_1^A \hat{\varphi}_1^A$ are computed and summarize the values

for the other state variables.

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

$$\exists n \in A.\exists m \in A.\ (\gamma(\hat{I}_1) \wedge g_1^n) \Rightarrow \varphi_1^m[asgn_1^n(\bar{x})/\bar{x}]$$

It is possible to pick a node n in $\gamma(A)$ such that φ_1^m. Suppose n is not the last node in $\gamma(A)$ and m is the next node after n in $\gamma(A)$, then $asgn_1^n(\bar{x})$ moves P_1 from node n to node m. In this case, if node m is empty before appending P_1 then φ_1^m is true of m after $asgn_1^n(\bar{x})/\bar{x}$.

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

$$\exists n \in A.\exists m \in A.(\gamma(\hat{I}_1) \wedge g_1^n \Rightarrow \neg\varphi_1^m[asgn_1^n(\bar{x})/\bar{x}]$$

Which is also true. If we pick n and m to be the same node in $\gamma(A)$, then after $[asgn_1^n(\bar{x})/\bar{x}]$ P_1 is no longer in m so $\exists m \in A.\neg\varphi_1^m[asgn_1^n(\bar{x})/\bar{x}]$ and we add $\hat{\vartheta}_1^A\neg\hat{\varphi}_1^A$ to $\hat{\tau}_1^A(\hat{I}_1)$.

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

$$\exists n \in A.\neg\exists m \in A.(\gamma(\hat{I}_1) \wedge g_1^n \Rightarrow \varphi_1^m[asgn_1^n(\bar{x})/\bar{x}]$$

Which is true if we pick n to be the last node in $\gamma(A)$. In this case, $[asgn_1^n(\bar{x})/\bar{x}]$ moves P_1 out of the last node in $\gamma(A)$ so that $\neg\exists m \in A.\varphi_1^m[asgn_1^n(\bar{x})/\bar{x}]$. So we add $\neg\hat{\vartheta}_1^A\hat{\varphi}_1^A$ to $\hat{\tau}_1^A(\hat{I}_1)$.

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

$$\exists n \in A.\neg\exists m \in A.(\gamma(\hat{I}_1) \wedge g_1^n \Rightarrow \neg\varphi_1^m[asgn_1^n(\bar{x})/\bar{x}]$$

Which is not true because P_1 is in at most one node in $\gamma(A)$ before and after $[asgn_1^n(\bar{x})/\bar{x}]$ 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 $\hat{\vartheta}_1^A\hat{\varphi}_1^A$, $\hat{\vartheta}_1^A\neg\hat{\varphi}_1^A$ and $\neg\hat{\vartheta}_1^A\hat{\varphi}_1^A$ are in $\hat{\tau}_1^A(\hat{I}_1)$. We have now computed the values of $\hat{\vartheta}_1$ and $\hat{\varphi}_1$ for path A. We next perform a similar analysis for paths B and C.

- For path B, only $\neg\hat{\vartheta}_1^B\hat{\varphi}_1^B$ is in $\hat{\tau}_1^A(\hat{I}_1)$. This is because $[asgn_1^n(\bar{x})/\bar{x}]$ 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 τ_1^C are in $\hat{\tau}_1^A(\hat{I}_1)$: $\hat{\vartheta}_1^C\hat{\varphi}_1^C$, $\hat{\vartheta}_1^C\neg\hat{\varphi}_1^C$ and $\neg\hat{\vartheta}_1^C\hat{\varphi}_1^C$. These terms are included because $[asgn_1^n(\bar{x})/\bar{x}]$ 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 φ_0 and φ_2 for bridges in paths A, B and C need to be included in $\hat{\tau}_1^A(\hat{I}_1)$.

- For predicate φ_0, $\hat{\tau}_1^A(\hat{I}_1)$ has no effect on φ_0 because $[asgn_1^n(\bar{x})/\bar{x}]$ can not violate the invariant. Suppose $[asgn_1^n(\bar{x})/\bar{x}]$ moves $(1,3)$ into the head of agent 3. Then if φ_0^3 was true before $[asgn_1^n(\bar{x})/\bar{x}]$ then φ_0^3 is still true after $[asgn_1^n(\bar{x})/\bar{x}]$. Suppose $[asgn_1^n(\bar{x})/\bar{x}]$ does not move $(1,3)$ into the head of agent 3. Similarly, if φ_0^3 was true before $[asgn_1^n(\bar{x})/\bar{x}]$ then φ_0^3 is still true after $[asgn_1^n(\bar{x})/\bar{x}]$.
- For predicate φ_2, $\hat{\tau}_1^A(\hat{I}_1)$ has no effect on φ_2 either. This is because $[asgn_1^n(\bar{x})/\bar{x}]$ does not affect message $(2,3)$. While $[asgn_1^n(\bar{x})/\bar{x}]$ does move message $(1,3)$, the location of message $(2,3)$ has no bearing on the value of φ_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:

$$\begin{aligned}\hat{\tau}_1^A(\hat{I}_1) = \ & \neg\hat{\vartheta}_0^A\neg\hat{\varphi}_0^A \wedge \neg\hat{\vartheta}_0^B\neg\hat{\varphi}_0^B \wedge \neg\hat{\vartheta}_0^C\neg\hat{\varphi}_0^C \wedge \\ & (\hat{\vartheta}_1^A\hat{\varphi}_1^A \vee \hat{\vartheta}_1^A\neg\hat{\varphi}_1^A \vee \neg\hat{\vartheta}_1^A\hat{\varphi}_1^A) \wedge \\ & \neg\hat{\vartheta}_1^B\hat{\varphi}_1^B \wedge \\ & (\hat{\vartheta}_1^C\hat{\varphi}_1^C \vee \hat{\vartheta}_1^C\neg\hat{\varphi}_1^C \vee \neg\hat{\vartheta}_1^C\hat{\varphi}_1^C) \wedge \\ & \neg\hat{\vartheta}_2^A\hat{\varphi}_2^A \wedge \hat{\vartheta}_2^B\hat{\varphi}_2^B \wedge \neg\hat{\vartheta}_2^C\hat{\varphi}_2^C\end{aligned}$$

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

$$\gamma(\hat{I}_1) \Rightarrow \forall n \in B.\neg g_1^n \text{ and } \gamma(\hat{I}_1) \Rightarrow \forall n \in C.\neg g_1^n$$

Both of these implications are true because $\neg\hat{\vartheta}_1^B\hat{\varphi}_1^B$ and $\neg\hat{\vartheta}_1^C\hat{\varphi}_1^C$ are in $\hat{I}_1$. For $\hat{\tau}_1^B(\hat{I}_1)$, we then have that

$$\neg\exists m \in B.\varphi_1^m \wedge \gamma(\hat{I}_1) \Rightarrow \forall n \in C.\neg g_1^n$$

and $\varphi_1^m = g_1^m$. Because no nodes in paths B and C satisfy the guard on τ_1 in state $\hat{I}_1$, 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 τ_1 on all network nodes of all concrete states represented by $\hat{I}_1$, we now analyze the effects of τ_2 on all nodes in all states represented by $\hat{I}_1$. The effects of τ_2 are similar to the effects of τ_1 except τ_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

$$\gamma(\hat{I}_2) \Rightarrow \forall n \in B.\neg g_2^n$$

Since $\hat{\vartheta}_2^B\hat{\varphi}_2^B$ is in $\hat{I}_1$, we have that

$$\exists m \in B.\varphi_2^m \wedge \gamma(\hat{I}_2) \Rightarrow \forall n \in B.\neg g_2^n$$

which is false since $\varphi_2 = g_2$. 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 $\hat{\vartheta}_0^C\hat{\varphi}_0^C$ because it will yield a violation of the form $\hat{\vartheta}_0^C\neg\hat{\varphi}_0^C$. To include $\hat{\vartheta}_0^C\neg\hat{\varphi}_0^C$ in the next state of $\hat{\tau}_2^B(\hat{I}_1)$ we need to prove that

$$\exists n \in B.\exists m \in C.(\gamma(\hat{I}_1) \wedge g_2^n \Rightarrow \neg\varphi_0^m[asgn_2^n(\bar{x})/\bar{x}]$$

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 $[asgn_2^n(\bar{x})/\bar{x}]$ moves message $(2,3)$ into the head of the queue in m. Bridge m now violates

$$x^3.q[0] = (2,3) \wedge (i = 3) \Rightarrow x^i.q[0] = ((1,3) \vee empty)$$

so we add $\hat{\vartheta}_0^C\neg\hat{\varphi}_0^C$ to $\hat{\tau}_2^B(\hat{I}_1)$ and find that a node exists in path C which violates φ_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_p 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 "paper-ware." 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 ∀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

[AK86] Krzysztof R. Apt and Dexter C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, May 1986.

[BJNT00] Ahmed Bouajjani, Bengt Johnsson, Marcus Nilsson, and Tayssir Touilli. Regular model checking. In Emerson and Sistla [ES00], pages 403–418.

[CGJ95] E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In *6th International Conference on Concurrency Theory (CONCUR'95)*, Philadelphia, PA, August 1995.

[DDP99] Satyaki Das, David L. Dill, and Seungjoon Park. Experience with predicate abstraction. In Halbwachs and Peled [HP99].

[ES00] E. Allen Emerson and A. Prasad Sistla, editors. *Computer-Aided Verification, CAV '00*, volume 1855 of *Lecture Notes in Computer Science*, Chicago, IL, July 2000. Springer-Verlag.

[Gru97] Orna Grumburg, editor. *Computer-Aided Verification, CAV '97*, volume 1254 of *Lecture Notes in Computer Science*, Haifa, Israel, June 1997. Springer-Verlag.

[GS97] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with PVS. In Grumburg [Gru97].

[HP99] Nicolas Halbwachs and Doron Peled, editors. *Computer-Aided Verification, CAV '99*, volume 1633 of *Lecture Notes in Computer Science*, Trento, Italy, July 1999. Springer-Verlag.

[HRW92] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, Netherlands, 1992.

[JG00] Michael Jones and Ganesh Gopalakrishnan. Verifying transaction ordering properties in unbounded bus networks through combined deductive/algorithmic methods. In *Formal Methods in Computer-Aided Design: FMCAD'00*, November 2000. To appear.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In Grumburg [Gru97].

[LS97] David Lesens and Hassen Saïdi. Automatic verification of parameterized networks of processes by abstraction. In *2nd International Workshop on the Verification of Infinite State Systems: INFINITY'97*, July 1997.

[MHJG00] Abdel Mokkedem, Ravi Hosabettu, Michael D. Jones, and Ganesh Gopalakrishnan. Formalization and proof of a solution to the PCI 2.1 bus transaction ordering problem. *Formal Methods in Systems Design*, 16(1):93–119, January 2000.

[PCI95] PCISIG. PCI Special Interest Group–PCI Local Bus Specification, Revision 2.1, June 1995.

[PS00] Amir Pnueli and Elad Shahar. Liveness and acceleration in parameterized verification. In Emerson and Sistla [ES00], pages 328–343.

[SG89] Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. In *Workshop on Automatic Verification Methods for Finite-State Systems*, volume 407 of *Lecture Notes in Computer Science*, Grenoble, France, June 1989. Springer-Verlag.

[SS99] Hassen Saïdi and N. Shankar. Abstract and model check while you prove. In Halbwachs and Peled [HP99].