Verifying transaction ordering properties in unbounded bus networks through combined deductive/algorithmic methods*

Michael Jones, Ganesh Gopalakrishnan  
50 S. Central Campus Dr. Rm. 3190  
Salt Lake City, UT 84112-9205  
mjones.ganesh@cs.utah.edu  
University of Utah

Abstract. Previously [MHG98,MHG00], we reported our efforts to verify the producer/consumer transaction ordering property for the PCI 2.1 protocol extended with local master IDs. Although our efforts were met with some success, we were unable to show that all execution traces of all acyclic PCI networks satisfy this transaction ordering property. In this paper, we present a verification technique based on network symmetry classes along with a manually derived abstraction that allows us to show, at the bus/bridge level, that all execution traces of all acyclic PCI networks satisfy the transaction ordering property. This now completed case study (modulo the validity of the axioms used to characterized the abstraction) suggests several avenues for further work in combining model-checking (algorithmic methods) and theorem-proving (deductive methods) in judicious ways to solve infinite-state verification problems at the bus/interconnect level. It is a concrete illustration of partitioning concerns where designers can specify bus protocols in an operational semantics (rule-based) style, invent abstractions, and carry out finite-state model-checking while verification experts can establish formal properties of the abstraction.

1 Introduction

Verifying transaction ordering properties in unbounded bus networks is of growing importance, given the prevalence of busses and systems on chip (SOC) interconnects. In this paper we present an approach that combines deductive and algorithmic methods to achieve exhaustive verification of safety properties over unbounded branching networks. An unbounded branching network is a class of network topologies in which there is no limit on the number of nodes that may be included in the network so long as the resulting network is both acyclic and completely connected. Our work is characterized by several features:

- It is a concrete illustration of partitioning concerns where designers can specify bus protocols in an operational (rule-based) style, invent

* Supported in part by NSF Grant No. CCR-9800928
abstractions, and carry out finite-state model-checking while verification experts can establish formal properties of the abstraction,

- It is a concrete case study involving the popular PCI 2.1 bus and the emerging Virtual Components Interface (VCI) SOC bus
- It is a concrete demonstration of how combined algorithmic/deductive methods may be deployed in practice.
- It provides a method to reason about non-trivial protocols over unbounded branching networks.

The subject of the case study was the PCI 2.1 protocol. The PCI 2.1 local bus protocol is an open standard that implements a message passing system over acyclic bus/bridge networks. The PCI standard supports two message types and allows message reordering and deletion in certain situations. Despite the possibility of reordering and deletion, the PCI protocol is intended to obey the producer/consumer (PC) transaction ordering property. Unfortunately, the published PCI standard violates the PC property due to a phenomenon called completion stealing [Cor96]. Completion stealing allows the consumer to read from the data address before the producer has written the new data value. In this case study, we verify a corrected version of the PCI protocol, which has been extended with local master IDs. Local master IDs were proposed as a solution to the completion stealing problem. For the purposes of this report, PCI will refer to the PCI protocol extended with local master IDs and PCI will refer to the original, erroneous, PCI protocol without local master IDs. We do not address cycle or physical level details of the PCI specification in this report.

Our overall approach to solving this verification problem was to use a manually derived, but formally justified, symmetry reduction on branching networks. This symmetry reduction reduced instances of the PC property over arbitrary PCI networks to one of four network symmetry classes\(^1\). Unfortunately, each of these reduced networks has an infinite number of states. A further modification of the PCI protocol was required to reduce the unbounded number of states in each reduced network, resulting in a protocol called PCI'. We then created a PVS proof in which we showed that the PCI protocol operating on bus/bridge networks is a trace inclusion refinement of the PCI' protocol operating on the networks from the symmetry classes. We then checked all execution traces of PCI' in all four reduced networks in just under 2 minutes of execution time using the Mur\(\ddot{o}\) model checker. No violations were found in model checking. The refinement proof taken together with the model checking results allow us to conclude that all traces of all PCI networks satisfy the PC property.

After exhaustively verifying the PC property for PCI, we turned our attention to two related verification problems: verifying the PC for the original PCI protocol without local master IDs and verifying an aspect of PCI/VCI interaction. We modified the PCI' protocol to create a PCI" protocol such that PCI" is a trace inclusion refinement of PCI'. Using the same symmetry reduction as in the PCI" verification, we found a violation of the PC property in the PCI" protocol which could be traced back

\(^1\) A symmetry class is a kind of equivalence class in which the members of a class are identical under a well-defined set of transformations.
to completion stealing in the PCI protocol. We also verified that the PCI protocol satisfies an in-order transmission policy required for transmitting packets chains in the Virtual Sockets Interface Alliance (VSIA, see [AI90]) VCI on-chip bus standard. This behavior is important should the on-chip bus VCI and the PCI be connected in an SOC environment. The key contributions of this paper are:

- The exhaustive verification of the PCI property for all execution traces in all PCI networks at the bus/bridge level.
- An abstraction technique for reducing unbounded sets of branching networks to finite sets of reduced networks. This is done using a symmetry reduction that ignores bridge boundaries and is parameterized by the number of agents in the property being verified.
- An abstraction technique for handling the unbounded states in each reduced network created by the symmetry reduction in the previous contribution. The fact that each reduced network has an unbounded number of states arises from a mechanism used to acknowledge delayed transactions in the PCI protocol. The unbounded states are reduced based on an invariant identified using a PVS model of PCI.
- An illustration that a combination of model checking and theorem proving together with network symmetry reductions and rule-based notation is an effective approach for reasoning about non-trivial protocols over branching networks.

The importance of the PCI case study itself stems from PCI being an important open I/O standard which finds wide use in computer systems; and is slated for migration into next generation technologies—such as systems on chips. This work extends our previous PCI verification work by formalizing the refinement proof. The refinement proof allows us to state the transaction ordering verification problem as a model checking problem. We plan to develop a network protocol verification tool based on the techniques used in this case study.

In the next section we sketch pertinent details of the PCI protocol and PC transaction ordering property with the aim of presenting the specific verification problem addressed in this report. Section 3 contains an overview of our particular solution with a more detailed description of each solution step given in each of the subsections. In section 4 we give our verification results for PCI, without local master IDs and the VSIA packet chain ordering problem. Section 5 contains related work and section 6 summarizes this our observations relative to this case study.

2 Problem Definition

In this section, we sketch an overview of the PCI protocol and network topology and describe the PC transaction ordering property specified. A PCI network is an acyclic hyper-graph of agents and bridges connected by busses such that there exists a unique path between any two agents. Agents are connected to one local bus while bus bridges connect ("bridge") two local busses. An example of a PCI network appears in the left half of Figure 3. As shown in the figure, agents and bridges each
contain two queues; one in each direction. The opposite queue of the top
queue in a bridge is the bottom queue (pointing in the opposite direction)
in the same bridge, and vice versa.
A PCI network supports two types of transactions: posted and delayed.
Posted transactions are unacknowledged transactions that can be nei-
ther deleted nor bypassed. Delayed transactions are acknowledged trans-
actions that can be bypassed at any time by other delayed or posted
transactions. A committed delayed transaction is a delayed transaction
that has been attempted, but not necessarily latched, on the local bus.
Delayed transactions may be dropped unless they have been committed.
Delayed transactions leave a trail of committed copies of themselves in
every bridge through which they pass. The response to a delayed trans-
action is called its completion. Completions travel back from target to
source following the trail of copies of the matching delayed transaction.
Completions can be dropped and bypassed—except for delayed write
completions, which cannot be passed. A posted transaction is consid-
ered complete at a bridge or agent as soon as it has been issued on the
local bus. Delayed transactions are considered complete at a bridge or
agent only when the completion has returned to that bridge or agent.
The rules for bypassing and dropping transactions are intended to pre-
vent deadlock while preserving the transaction ordering required by the
PCI property.
The PCI specification requires that PCI bus/bridge networks have the
PCI transaction ordering property. For the purposes of PCI, the PCI prop-
erty is stated as follows.
If the following preconditions are satisfied:
(1) An agent, the producer (Prod), issues two write transactions: \( W_{Data} \)
   (a posted or delayed write transaction to the address \( Data \)) fol-
   lowed by \( W_{Flag} \) (a posted or delayed write transaction to the address
   \( Flag \)).
(2) An agent, the consumer \( Cons \) issues two Delayed Read Request
   transactions \( R_{Flag} \) followed by \( R_{Data} \),
(3) \( W_{Flag} \) is issued on the originating bus after the completion of \( W_{Data} \)
on the originating bus,
(4) \( R_{Data} \) is committed on the originating bus after the completion of
   \( R_{Flag} \) on the originating bus,
(5) \( R_{Flag} \) is completed on the destination bus after \( W_{Flag} \) completes on
   the destination bus.
Then, assuming no other agents write to the data address, the value
returned by \( Cons \) \( R_{data} \) is the value written by \( Prod \) \( W_{data} \).
The central problem in this case study then is to show that all execution
traces of all PCI networks satisfy the PC property.

3 Solution

Our solution to the problem of exhaustively verifying the PC transac-
tion ordering property is outlined in figure 1. The solution can be divided
into three steps and each step is labeled 1, 2 or 3 in the figure: First, we use
a manually discovered branching network symmetry reduction on PCI bus/bridge networks which requires us to transform the PCI protocol into PCI' (see section 3.1). Second, we use an interactive higher-order logic theorem prover to prove that the PCI protocol on PCI networks is a trace inclusion refinement of the PCI' protocol on the reduced networks (section 3.2). Third, we use an explicit state model checker to show that all execution traces on the reduced networks satisfy the PC property (section 3.3).

The goal of the branching network symmetry reduction is to map arbitrary instances of the PC property over a PCI network to a finite number of instances over a finite number of reduced networks. In [MHG98] we showed that we could map all instances of the PC property on PCI to one of four reduced networks. This structural symmetry reduction allows us to cover all instances of the PC property over all PCI networks by checking the execution traces of four abstract networks. The symmetry reduction applies to all unbounded branching networks, not just PCI networks. Using the symmetry reduction requires us to modify the PCI protocol so that each reduced network has a finite number of states; resulting in the PCI' protocol.

Our goal in showing that PCI is a trace inclusion refinement of PCI' was to formally establish that safety properties of PCI' hold for PCI as well. This allows us to apply model checking results from PCI' to PCI. Proving
trace inclusion refinement requires showing that for every concrete trace, there exists an abstract trace such that each state in the abstract trace is equal to the abstraction of the corresponding concrete state [AL91]. Trace inclusion is conservative with respect to safety properties. This means that properties violated by PCI' are not necessarily violated by PCI. In the context of PCI, for each concrete trace $\sigma$, we need to show that there exists an abstract trace, $\sigma_A$, in PCI', such that the following relationship holds:

$$\forall \sigma \in \text{PCI}. \{ \sigma = (A_0, C_0), (A_1, C_1), (A_2, C_2) \ldots (A_n, C_n) \}$$

$$\Rightarrow \exists \sigma_A \in \text{PCI}'. \sigma_A = (A_0, A_1, A_2 \ldots A_n)$$

in which for every $A_i$ in $\sigma_A$, $A_i = \text{abs}(C_i)$ for some abstraction function $\text{abs}$. The proof is outlined in figure 2. As shown in the figure, the relationship will be shown by induction on the length of $\sigma$. The operational semantics of each protocol were defined inductively using a set of inference rules. A set of inference rules, one per PCI transition, describe the operational semantics of each PCI and PCI' transition in terms of the transition's effect on the system state. The set of reachable states is inductively defined, starting from an initial state, as the parallel composition of the inference rules. An inference rule can be applied to a reachable state to create a new reachable state if and only if the rule's preconditions are satisfied by the current reachable state. A state is reachable if and only if it can be derived from the initial state by a sequence of applications of the inference rules. All of the rules needed to define the operational semantics of PCI and PCI' can be found at [Jon00]. The inference rules facilitate the inductive refinement proof by suggesting a natural way to perform a case split on the construction of the next state. The rules were then used to split the inductive step by showing that the abstraction of every application of every PCI rule, labeled $\delta_i$, has a corresponding application of a PCI' rule, labeled $\delta_A$.

Our goal in model checking PCI' was to show that every trace in every reduced network satisfies the PC property. The inference rules describing the operational semantics of PCI' were used to define a model of PCI' for use in the Mur$\phi$ model checker [ID96]. We checked all PCI' traces of all four reduced networks in just under 2 minutes. No violations of the PC property were found in PCI' which, due to the refinement relation, means that all traces of the PCI protocol on all PCI networks also do not violate the PC property.

The next three sections are each devoted to explaining the network symmetry reduction, refinement proof and model checking results in more detail.

### 3.1 Network Symmetry Reduction

The symmetry reduction allows us to map instances of the PC property to one of four reduced networks, but requires us to modify the PCI protocol to keep the number of states in each reduced network finite. Consider the problem of showing that $n$ agents in an unbounded branching network satisfy a certain property. In the context of PCI there may also
be arbitrarily many other agents which might affect the property being verified; in addition, the paths between any two agents may contain arbitrarily many bridges and busses. Despite the fact that each network has a (large) finite number of states, there are an unbounded number of networks. One way to reduce this problem over an unbounded number of networks to a bounded number of networks is to consider the symmetry classes induced by ignoring agents and paths incidental to the property being verified and to ignore the lengths of the paths between agents central to the property being verified.

Figure 3 contains an example of such a symmetry reduction on an instance of the PC on a PCI network. The network on the left side of the figure contains an instance of the PC property defined on a network with seven agents and two bridges. The producer, consumer, data and flag agents are labeled P, C, D and F while the other agents are all labeled with A's. The parts of the network on the left side of the figure that contribute to the reduced network have been circled. The reduced network appears on the right side of the figure. In the reduced network, all agents incidental to the PC property, labeled with A's, are ignored. In addition, the path between bridges 1 and 2 in the center of the network on the left have been coalesced into path B in the network on the right. The transactions in the queues of bridges 1 and 2 are concatenated and placed in path B. One can add arbitrarily many busses, bridges and agents to the network on the left and it will still reduce to the same network on the right.

Despite the fact that we are now concerned with only four reduced networks, the PCI protocol over each of these reduced networks has an unbounded number of states. Recall that the contents of each bridge in a path are concatenated to form the contents of a path. Although each bridge contains a bounded number of transactions, there may be arbitrarily many bridges in a path. This means there may be arbitrarily many transactions in a path. We can reduce the number of transactions in a path by ignoring transactions other than those required by the property being verified. However, in the case of PCI, this still allows paths with an unbounded number of transactions. Recall that as delayed transactions travel between source and destination, they leave a trail of committed copies of themselves in each bridge through which they pass. Each of these committed copies, one per bridge, are then concatenated and placed in the corresponding path in the reduced network—again
leading to an unbounded number of states in the reduced network. We eliminate this source of unboundedness by keeping only the newest copy of a committed delayed transaction in the state of the reduced network. This reduction is justified by a PCI invariant and required several modifications to the PCI' protocol. The modifications to the PCI' protocol were required so that the upcoming PVS proof that PCI is a refinement of PCI' would go through.

3.2 PVS Refinement Proof

The refinement proof was carried out in the PVS theorem prover using three theories which described PCI, PCI' and the abstraction. We first describe each of the three theories then discuss the PVS refinement proof. All three theories and the complete refinement proof can be found at the following URL [Jon00].

**PCI Theory** The theory describing the concrete PCI protocol contained ten inference rules and several supporting definitions. Each of the ten rules describe a PCI transition and were adapted from [CS97,PCI95]. A set of 10 inference rules which describe the operational semantics of PCI can also be found at [Jon00].

Given a reachable state, a rule constructs a new reachable state if the preconditions are met. The preconditions to each state involve at most a bridge (or agent) and its adjacent bridge (or agent). If the preconditions are met, the next reachable state is constructed by modifying the contents of a bridge (or agent) and its adjacent bridge (or agent). The limited scope of each rule allows us to apply the same set of inductive rules to networks with different topologies. This is because each rule depends only on the connection between two entities, rather than the topology of the entire network.

The PCI theory contains 551 lines of PVS, including comments. The theory is built on top of a theory of unbounded finite sequences which are used to model the queues. Two rules cover the movement of posted transactions while the remaining eight cover the movement of delayed transactions and their completions. Starting from a network in its initial state, the entire set of reachable states for the network can be computed by repeatedly applying rules whenever and wherever their preconditions are satisfied.

**PCI' Theory** The PCI' rules were designed so that they would mimic every effect of the PCI rules on the reduced networks while preserving as many properties of PCI as possible. If the PCI' rules mimic every effect of the PCI rules then we can show that PCI is a refinement of PCI'. Suppose we chose the PCI' rules to be the CHAOS rule-set, in which all actions are always allowed. In this case, we can certainly show that PCI is a refinement of CHAOS, but we can not show that CHAOS has any useful properties. By carefully designing the PCI' protocol, we create a protocol which is refined by PCI but which also has useful properties. A set of inference rules describing the operational semantics of the PCI' protocol can also be found at [Jon00].
Fig. 4. Example of ambiguity in reduced network paths.

The theory describing the PCI' protocol contained fifteen rules and several supporting definitions. The PCI' theory contains more rules than the PCI theory because additional rules are needed to compensate for the network information lost in the abstraction. The additional rules in the PCI' theory stem from the loss of queue boundaries and loss of committed copies of delayed transactions under the abstraction. Figure 4 illustrates the ambiguity resulting from the loss of queue boundaries. Both network fragments on the left, N1 and N2, contain a single transaction, T1 and T2. A path boundary is indicated by the dashed line. Despite the fact that T2 has reached the final position in the path, since no transactions appear ahead of T1 in N1: both T1 and T2 map to the head of the paths P1 and P2 under the abstraction. This would not be a problem, except the new transactions created by latching T1 and T2 in N1 and N2 appear in different locations in P1 and P2. In P1, the new transaction would appear in front of T1' and in P2 the new transaction would appear at tail of the next path after P2. As a result, PCI' requires two versions of certain latching rules to cover both cases. Similar instances of ambiguity involving suppressed committed copies of delayed transactions necessitate additional rules in the PCI' model.

The PCI' theory is 507 lines long, including comments. The PCI' theory uses the same theory of finite sequences to represent paths. Of the fifteen rules, three cover the movement of posted transactions and the other twelve cover the movement of delayed transactions and their completions.

Abstraction Theory. The abstraction theory describes the abstraction function that relates states in PCI to states in PCI'. The abstraction function performs the network symmetry reduction and eliminates transactions (as discussed in section 3.1) so that each reduced network has a finite number of states. Rather than define the abstraction using a set of definitions and prove lemmas about the abstraction for use in the refinement proof, we chose to state needed properties about the abstraction as axioms and use the axioms directly in our refinement proof. We chose an axiomatic rather than a definitional theory to save the time required to prove each of the abstraction lemmas and instead focus on the re-
 refinement proof. A total of 51 axioms about the abstraction were used in the refinement proof. These axioms describe the effects of the network symmetry reduction, predicates which can be inferred about a PCI state from a PCI state and the effects of inserting and deleting transactions. Thirty of the axioms have been shown to be non-contradictory. The remaining 21 are believed to be non-contradictory, but the PVS proof is still under development. The 21 un-validated axioms describe the effects of inserting and deleting transactions.

The main source of complexity in the remaining un-validated axioms is the appearance and disappearance of committed delayed transactions in the states of PCI'. For example, if a new transaction is inserted into a path, then the significant committed delayed transactions already present in that path may or may not disappear depending on the type and location of the newly inserted transaction. This behavior under deleting and inserting transactions made defining axioms which describe the effects of insertion and deletion difficult. If PCI did not include a notion of “leaving trails of transactions” the axiomatization of the abstraction would have been smaller and simpler (but not as interesting).

**Refinement Proof** The refinement proof was done by showing that every application of every concrete rule corresponds to some application of an abstract rule. More specifically, we show that for all states s, the abstraction of every state created by every application of every PCI rule to state s is equal to some state created by the application of some PCI' rule to the abstraction of s. Using the inference rules from our theory of PCI, we divided the refinement proof into ten cases—one for each rule.

The PVS proof was developed by a single experienced PVS user in about one month of effort. The final refinement proof required approximately 1000 proof commands (determined by taking the number of lines in the PVS proof file and dividing by two to account for the pretty-printing of right parentheses). While the proof required a significant amount of effort, the refinement proof can be reused for exhaustively verifying other properties of the PCI protocol, as will be shown in section 4.

### 3.3 Model Checking

We wrote Murφ models of the PCI' protocol over the four network classes. While we intend for the Murφ and PVS models to have the same meaning, there is no formal relationship between the Murφ models and the PVS model of the PCI' protocol. The Murφ modeling language allows natural expression of a protocol defined using rules. The Murφ models were an average of 670 lines long (including comments). The Murφ code for the PCI' d-commit rule is given below. The rule is part of a larger rule-set in which the subscripts i and j range over all positions in all paths in the network. Murφ rule-sets provide a convenient notation for quantifying rules. In the Murφ model, transactions are written $dr$, $fr$, $drc$ and $frc$ to denote “data read,” and “flag read” transactions. A c at the end of a transaction indicates that the transaction has been committed.
Rule "d Commit 1"
( uncommitted (trans) &
 (trans - dr) -> (!member (path, fr) | member (path, frc)))
=>
begin delete_trans (path, commit (trans)); -- delete t_t
  network[i][j] := commit (trans);
end;

The precondition checks that the transaction is uncommitted, the additional predicate on dr encodes one of the preconditions to the PC property. The action, given after the "=>", replaces the transaction with its committed form.

<table>
<thead>
<tr>
<th>Network</th>
<th>CPU Time</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>35.35 sec</td>
<td>1614</td>
</tr>
<tr>
<td>B</td>
<td>18.68 sec</td>
<td>914</td>
</tr>
<tr>
<td>C</td>
<td>12.56 sec</td>
<td>648</td>
</tr>
<tr>
<td>D</td>
<td>51.20 sec</td>
<td>2690</td>
</tr>
<tr>
<td>Total</td>
<td>117.79 sec</td>
<td>5866</td>
</tr>
</tbody>
</table>

Fig. 5. Model checking results for PC property on PCI'.

We defined two pieces of auxiliary state to encode the PC property as a safety property. The auxiliary state stored the completion order of the read and write transactions at their respective destination agents. Using the completion orders stored in the auxiliary state, we can state the PC property as a safety property which reads:

If the data read transaction has completed at the data address, then either flag write occurred after the flag read, or the flag read has not completed, or the data read occurred after the data write.

This safety property is stated as an invariant of the Murϕ model of PCI' and checked using explicit state enumeration. The time and number of states required to check each reduced network is given in table 3.3. No violations were found in any of the networks. The Murϕ models were developed and verified in about one week of effort by a new Murϕ user (including time to install and learn Murϕ).

4 Verifying PC for PCI and an SOC Property

After using the abstraction to verify the PC property for PCI extended with local master IDs, we decided to verify the PCL as originally defined without local master IDs. We expected that this experiment would reveal a violation of the PC property due to completion stealing. We also verified the PCI protocol for another transaction ordering property related to packet chains used in the VSI for SOC applications. We now describe each experiment in turn.
4.1 Verifying PC on PCI\textsubscript{r} without Local Master IDs

We created a new reduced model, PCI\textsubscript{r}, so that PCI is a trace inclusion refinement of PCI\textsubscript{r}. Removing local master IDs in PCI\textsubscript{r} meant that all completions look the same and we could no longer distinguish between related and unrelated completions relative to the property being verified in the PCI\textsubscript{r} model. This meant that completions from other agents removed by the abstraction must be allowed to appear and disappear at any location at any time in the PCI\textsubscript{r} model. We did build a PVS proof showing that PCI refines PCI\textsubscript{r} since the changes from PCI to PCI\textsubscript{r} are minor.

We created a Mur\phi model of PCI\textsubscript{r} with a restriction that committed transactions could only appear and disappear at the head or tail of a path (to limit state explosion), and checked the model using breadth first state traversal. A violation of the PC invariant was found at a depth of 16 states after checking 38,436 states in just under ten minutes. The resulting error trace correlated to a PCI\textsubscript{r} error trace involving completion stealing, as reported by Corella [Cor96]. In general, trace inclusion refinement does not require that violations in the PCI\textsubscript{r} model imply violations in the PCI model, however, in this case we were lucky. The Mur\phi model of PCI\textsubscript{r} was created by modifying one of the PCI\textsubscript{r} models.

4.2 PCI Transaction Ordering for SOC

Next, we verified the property that “only posted transactions are always received in the order they were sent.” This in-order property is useful for packet chains transmitted between two virtual components using the VCI on-chip bus transported over a PCI network. The VCI on-chip bus standard requires that packet chains transmitted between VCI components must be received in the order they were transmitted regardless of what communication medium is used to transport them.

In the case of the in-order property, there are only two agents in the property: the sender and receiver. Since the in-order property uses only two agents, there is only one network symmetry class to consider (rather than four as required for the PC property). We re-used the Mur\phi model of PCI\textsubscript{r} used in the PC verification to check all traces of the abstract network and found (as expected) that only posted transactions, or posted transactions followed by a single delayed transaction, are received in order over PCI networks. Order in not guaranteed for any other combination of PCI transactions. Re-using the PC Mur\phi model to check the in-order property required changing the invariant and the topology of the network. No other changes to the refinement proof or model were needed.

5 Related Work

We compare our work to two other bodies of work: other PCI verification work and other complete verification methods for parameterized or unbounded systems. Our work is done at the same level of abstraction as
the liveness work done by Corella [CSZ97]. However, we consider transaction ordering rather than liveness and we use a combination of theorem proving and model checking rather than a manual proof. The PCI specification was used in a limited sense in a case study to demonstrate a BDD abstraction technique by Clarke [CJLW99]. An inconsistency in a state machine in the PCI specification was found by them in a network consisting of a master, an agent and a bus. In our PCI work, we consider all PCI networks and we reason about transaction ordering rather than message response. Finally, the work reported here extends our own previous work [MHG98, MHJG00] in that we have now exhaustively verified all execution traces of all PCI networks. This exhaustive verification depends on an abstraction which was hinted at in [MHJG00], but which we present here in detail and justify using a mechanical refinement proof. Although many techniques for reasoning about parameterized or unbounded systems exist, we found none as well-suited for reasoning about transaction ordering properties of non-trivial protocols over unbounded branching networks. Das [DDP99] presents a method which requires the user to supply a set of predicates to split the state space into abstract states which are constructed iteratively using BDDs to represent the reachable states. Abdulla [AAB+99] addresses the problem of verifying systems operating on infinite data structures using a combination of automatically generated abstractions and symbolic reachability analysis. Our method is most similar in aim to that presented by Kesten et al using predicate transformers for reasoning about properties on unbounded branching networks [KMM+97] applied to the Futurebus+ protocol. At the bus/bridge level, Futurebus+ and PCI are similar in that they allow arbitrary branching networks of processes. Predicate transformers over regular string expressions were used to reason about single-bus instances of the Futurebus+ protocol. While predicate transformers over regular tree expressions can be used to reason about properties over arbitrary branching networks, to the best of our knowledge, such an extension has not yet been reported. We have presented an abstraction technique that handles multiple bus instances of the PCI protocol.

Our technique for reasoning about networks is also similar to that of Bhargavan et al [BOG00] in which a combination of HOL theorem proving and SPIN model checking is used to reason about properties of several routing information protocols including the Bellman-Ford algorithm and a protocol under development for ad hoc mobile networks. Bhargavan’s work does not include the requirement of acyclic networks but is specifically targeted at routing information protocols rather than message passing protocols.

6 Concluding Remarks

Based on our experiences in this case study, we offer the following observations about the application of formal reasoning to the verification of non-trivial protocols over branching networks:

- Theorem proving or model checking applied directly to the problem is respectively too difficult or too complex. We found that iden-
tifying and proving invariants using a direct theorem proving approach required a prohibitive amount of manual effort. Although model checking applied directly to PCI is not possible given the unbounded number of network topologies, as reported in [MHJG00], we found that even modest coverage of the large, but finite, execution space of several concrete networks using a model checker was beyond the capacity of current model checking tools.

- The abstraction based on network symmetry modulo path lengths was essential but required significant modifications to the PCI protocol. These modifications required a refinement proof to show that the modified protocol preserved the safety properties of the original protocol.
- Expressing the operational semantics of PCI and PCI' using rules provided a single convenient notation for both the refinement proof and model checking. We re-emphasize that a total of 10 rules, which can be printing on one page, were required to specify the behavior of the entire PCI protocol. In addition, translating the PVS theory of the PCI' rules used in the refinement proof into a set of rules for a Muré model required little effort. Creating a Muré model of PCI' from the PCI' rules required about one week of effort by a new Muré user.
- The current abstraction and refinement proof result in a model that can be exhaustively examined very quickly, however, deriving the abstraction and showing refinement required a great deal of manual effort.

Based on the above observations, we are pursuing general methods for reasoning about parameterized systems in acyclic branching topologies based on abstractions that are “correct by construction” in the sense that the resulting abstraction will not require a refinement proof. We anticipate that this abstraction technique will require more CPU time to check the reduced model, but will require less user time to show refinement. At present, we are investigating the use of predicate abstraction and regular expressions to represent the state of each component and nonisomorphic Steiner trees over labeled terminals to cover the network symmetry classes.

References


