Transaction ordering verification of the PCI 2.1 Protocol using Trace Inclusion Refinement

Michael Jones, Ganesh Gopalakrishnan

January 13, 2000

Abstract

We define an abstract model of PCI, called PCI_A, and show that PCI is a refinement of PCI_A based on trace inclusion. We then show that no traces of PCI_A violate the Producer/Consumer property using the murϕ explicit state ennumeration model checker. Given that PCI_A does not violate the Producer/Consumer property, we conclude that PCI does not violate the property either. This work is the first complete formal arguement that PCI satisfies the Producer/Consumer property.

1 Introduction

We first define a representation of PCI internal states then show how to map an internal PCI state into an observable external state using a function called α. Both the internal and external states of PCI_A are taken from the external state space. Next, we give a proof sketch for PCI $\sqsubseteq$ PCI_A. The complete pencil and paper proof can be found in [JHM99]. Finaly, we use an exhaustive state space search technique to verify that no traces of PCI_A violate the producer/consumer property. We compare this approach to verifying transaction ordering for PCI with other approaches we have taken.

2 Internal and External States

Informaly, a PCI network is a collection of busses, bridges and agents arranged in an acyclic network in which there is a path between any two agents. Agents contain incoming and outgoing queues for recieving or sending transactions on the network. Bridges sit between two busses and contain two queues with one queue for recieving or sending transactions from either bus. Transactions are initiated by a master agent and addressed to a destination agent. Routing is handled by a static routing table.

A PCI network supports two types of transactions: posted and delayed. Posted transactions are unacknowledged transactions that can never be deleted nor passed by other

transactions. Delayed transactions are acknowledged transactions that can be deleted in certain situations and can always be passed by other delayed or posted transactions. Delayed transactions leave a trail of copies of themselves in every bridge they pass through. The reponse to a delayed transactions is called its completion. Completions travel back from target to source following the trail of delayed transaction copies. Completions can be deleted in certain situations and can be passed--except for a completion for a write *** or is it read? transaction. A posted transaction is considered complete when issued while a delayed transaction is considered complete after is completion has returned.

The transaction reordering and deletion rules are intended to prevent deadlock while preserving the *producer/consumer*(producer/consumer) property. The producer/consumer property states that if an agent (the producer) issues a write to a data address followed by a write to a flag address and then another agent (the consumer) reads the flag address then reads the data address then the data returned by the consumer agent data read will be the new value written by the producer.

2.1 A Formal Model of PCI and producer/consumer

We now build a formal model of PCI networks which we will use to formally define producer/consumer an abstract model of PCI networks and a refinement map between them. We begin with transactions. The j-th transaction in queue i, t_{ij}, is a four tuple

$$t_{ij} = type(source, target, committted)$$

where *type* is one of P, D or C, *source* is the issuing source queue, *target* is the destination queue and *committed* is a boolean indicating whether a D type transaction is committed or not. In the case of completions, the source queue is the source of the matching delayed transaction and the target is the target of the matching delayed transaction. We also adopt the notation that for a delayed transaction t, t_f is the uncommitted form of t (that is, $t = D(s, t, f)$ for some source s and target t) and t_t is the committed form.

A queue is simply a finite list of transactions.

$$q_i = (t_{i1}, t_{i2} \ldots t_{in})$$

A bridge or agent contains two queues. For bridges, these are queues which source the bus on either side of the briddge. For agents, these are the source and destination queues. A bridge or agent numbered i contains queues q_i and q_i'. The opposite queue for q_i is q_i'. Through the remainder of this paper, queue and bridge will be used interchangably unless otherwise noted.

A path is a finite list of queues specified by the queues at each end of the path. For example, the path

$$p_{jk} = (q_j, q_k)$$

contains all queues between q_j and q_k.

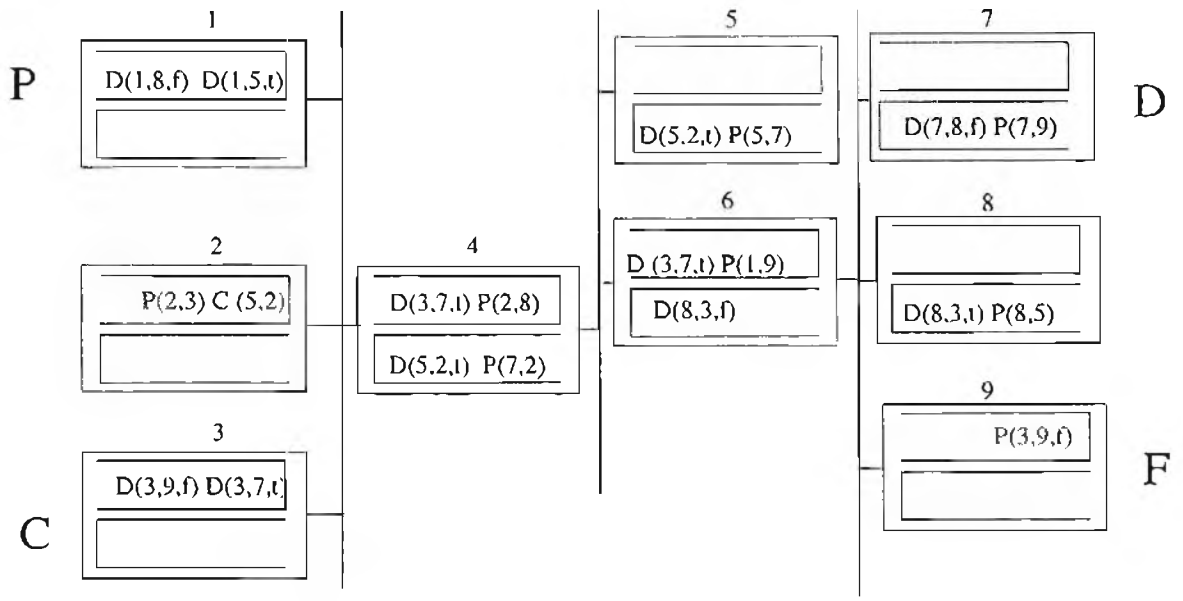

Figure 1: State of an example PCI network

For example, the state of the network shown in figure 1 has the following represention:

$$
\begin{aligned}
N = \quad & (q_1 = (D(1,8,f), D(1,5,t)), q_1' = (), \\
& q_2 = (P(2,3), C(5,2)), q_2' = (), \\
& q_3 = (D(3,9,f), D(3,7,t)), q_3' = (), \\
& q_4 = (D(3,7,t), P(2,8), q_4' = (D(5,2,t), P(7,2), \\
& q_5 = (), q_5' = (D(5,2,t), P(5,7)), \\
& q_6 = (D(3,7,t), P(1,9)), q_6' = (D(8,3,f)), \\
& q_7 = (), q_7' = (P(7,9), D(7,8,f)), \\
& q_8 = (), q_8' = (P(8,5), D(8,3,t)), \\
& q_9 = (D(3,9,f)), q_9' = ())
\end{aligned}
$$

and the path from queues 2 to 7 contains queues 2,4,6 and 7. We adopt the convention that the contents of a queue are listed left to right with the head of a queue at the right. In the figure, the agents labeled P, C, D and F will be the producer, consumer, data and flag agents and are used next in the explanation of the producer consumer property.

The producer/consumer property first requires identifying four agents: the producer, consumer, data and flag agents, abbreviated Pr, Co, Da and Fl respectively. The producer/consumer requires that, if:

1. Producer issues a write to data followed by a write to flag, and
2. Consumer issues a read to flag followed by a read to data, and
3. The data write completes at the Producer before the flag write is committed at the Producer, and
4. The Flag read completes at the Consumer before the Data read is committed at the Consumer, and

5. The Producer's write to Flag completes at the flag before the Consumer's read to Flag, and
6. No other agents write to the Data or Flag addresses betwen the time the Producer writes the data or flag and the time the Consumer reads the data or flag.

then the value written to Data by the Producer is returned by the Consumer's read to Data.

An *instance*, I, of the producer/consumer property is four agents and four transactions associated with an instantiation of the producer/consumer definiton in a PCI network. The agents in I are the producer, consumer, data and flag agents. The transactions in I are the data and flag read and write transactions. Since posted transactions are not acknowledged and do not return data, the data and flag writes can be either posted or delayed transactions while the data and flag reads must be delayed transactions. The *significant transactions* associated with an instance of the producer/consumer property are denoted as follows:

$$\begin{aligned} dw &= P(Pr, Da) \\ fw &= P(Pr, Fl) \\ fr &= D(Co, Fl, f) \\ frc &= D(Co, Fl, t) \\ cfr &= C(Co, Fl) \\ dr &= D(Co, Da, f) \\ drc &= D(Co, Da, t) \\ cdr &= C(Co, Da) \end{aligned}$$

In the remainder of this paper we consider only posted data and flag writes. The completion and commitment order imposed by precondition three trivialy satisfies the producer/consumer property if the data write is a delayed transactions. If the data write is a delayed transaction, then the data write is not complete at the producer until the cdr returns to the producer. In this case, the flag read is not allowed to commit at the producer until the after the data write completes at the producer. This means that the data write must complete at the data agent and return before the flag write is committed. Since the flag write must complete at the flag agent before the flag write (see precondition five), and the data read must not commit at the consumer until after the flag read completes at the consumer (see precondition four), the data read does not leave the consumer agent until after the data read has already completed at the data address.

We also ignore the case in which the flag write is a delayed transaction and the data write is a posted transaction. We ignore this case because it is subsumed by the case in which both are posted transactions. If the data write is a posted transaction, then the flag write may be committed by the producer as soon as the data write leaves the producer agent. This allows both the flag and data writes to travel through the network at the same time. However, if the data write is a posted transaction then the flag write can not pass the data write regardless of the type of the flag write. Ignoring the case in which the flag write is a delayed transaction does miss some reorderings between the flag read and write transactions. If the flag write is a delayed transaction, then the flag read and write can pass eachother. If the flag write is a posted transaction, the flag write may pass the

flag read but the flag read may never pass the flag write. We disregard these reorderings because precondition five requires that the flag write completes at the flag agent first in either case—regardless of any intermediate reorderings.

Finaly, the state of a PCI network will be augmented with two additional sets: a set of *issued significant transactions* S, and a set of *completed transactions* CT. Set S contains only the significant transactions that have already been issued for an instance of producer/consumer. The set CT contains pairs of transactions and agents, (t, a), indicating that transaction t has completed at agent a.

2.2 Formal Definition of an Abstract PCI

Our PCI abstraction, PCI_A, is parameterized by an instance, I, of the producer/consumer property as defined previously. A PCI_A network state contains a set of *significant paths*, M, which in turn contain signficant transactions. The significant paths are defined to be the paths connecting the Pr, Co, Da and Fl agents. All other paths in the network are considered *insignificant paths* and are discarded by the abstraction map. The significant paths are divided so that the common path (if any) between Pr,Co and Da or Fl is a seperate path. We have previously shown using our PVS model of PCI that the network in any instance of producer/consumer can be reduced to one of three sets of paths.

A path in PCI_A contains two finite lists of transactions—one in each direction. PCI queue boundaries are ignored in PCI_A paths. Only transactions from the set of *significant transactions* defined previously appear in PCI_A paths. All other transactions are abstracted away by our abstraction map. The statement of producer/consumer for PCI_A is the same as for PCI. We next define the abstraction mapping, α_I, for finding the abstract network state for a given concrete network state in an instance of the producer/consumer property. After defining the abstraction mapping, we apply α_I to the network state from Figure 1.

The abstraction function α_I, depends on the *transaction abstraction function* τ and the path abstraction function ψ.

$$\tau(t, M) = \begin{cases} t & \text{if } t \in T \text{ and, either newest-d}(t,M) \text{ or } t = P(\_,\_) \text{ or } t = D(\_,\_,f) \\ \epsilon & \text{otherwise} \end{cases}$$

Fuction $\tau(t, M)$ discards all transactions except the the significant transactions. Among significant transactions, all posted, uncommitted delayed transactions and completions are kept. Only the newest copy of a committed significant transaction in a given path is kept. Abstracting all but the newest committed transaction allows paths containing different numbers of queues to have the same abstract representation, but makes the refinement proof that follows more complicated. Keeping all copies of a committed transaction in a path would encode the number of bridges in a path as a delayed transaction leaves committed copies in every bridge.

The *path abstraction function*, ψ, abstracts insignificant brigdes on other paths and removes queue boundaries on significant paths. Function ψ also applied τ to the transactions on significant paths.

$$\psi(q_n, \tau, M) = \begin{cases} \text{M with insert } (q_n, m, \tau) & \text{if } q_n \in \text{path } m \text{ for some } m \in M \\ \epsilon & \text{otherwise} \end{cases}$$

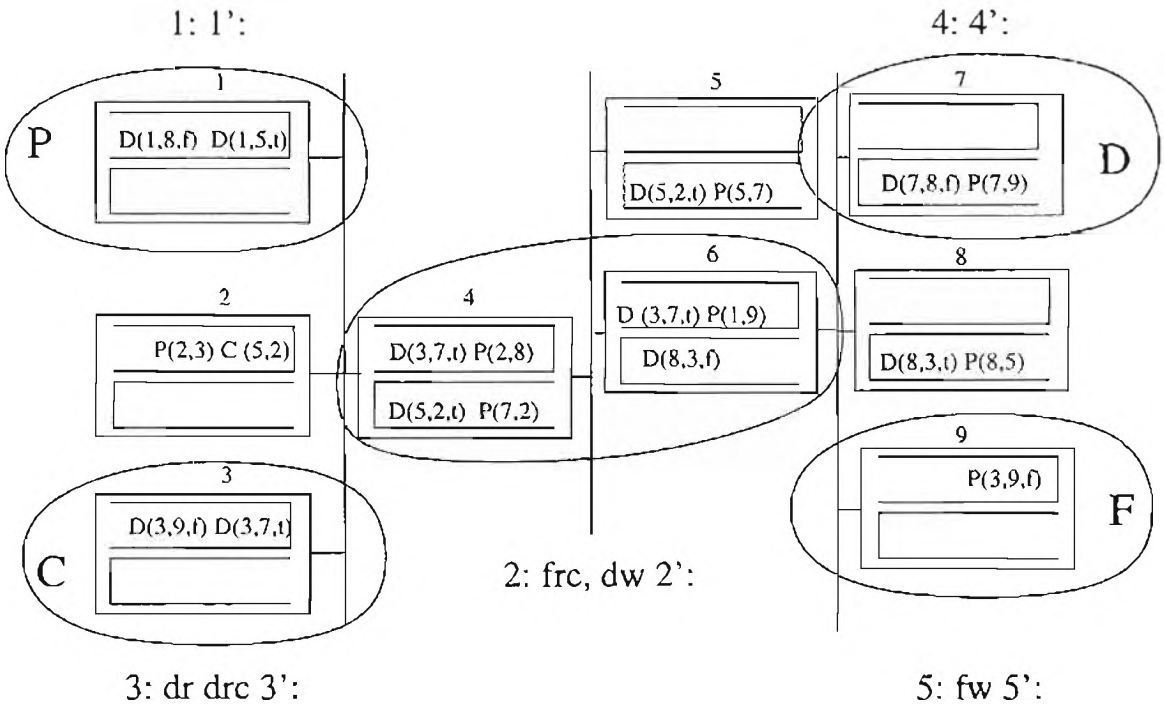

Figure 2: Abstracting an instance of the producer/consumer property in a PCI network

If queue q_n lies on some path m that is in the significant path set, M, then ψ inserts the significant transactions from q_n (as determined by τ) into path m in the abstract state. The fuction *insert* places the significant transactions from the queues in path m in order to form a list of significant transactions contained in all the queues in m.

We now define the network abstraction α_I in terms of ψ, τ and M_0. The network state M_0 is the *initial abstract network* consisting of paths containing no transactions. The network abstraction maps concrete networks, which are sets of queues, into abstract networks, which are sets of paths.

$$\alpha_I(\psi, \tau, M_0, N) = \psi(q_1, \tau, \psi(q_2, \tau, ...\psi(q_n, \tau, M_0)))$$

An abstract network state also includes an *abstract issued significant transactions set* S_α and an *abstract completed transactions set* CT_α. S_α is a copy of the issued transactions, S, from the concrete state. Set CT_α is created by maping τ over CT to keep only the significant completed transactions.

We use the network in figure 1 to provide an example of the abstraction function applied to an instance of the producer/consumer property to a concrete network state. The concrete network is redrawn in figure 2 to highlight the abstract paths. Recall that the abstraction function depends on the location of the producer, consumer, data and flag agents. For the sake of illustration, we choose q_1 as the producer, q_3 as the consumer, q_9 as the data and q_7 as the flag. Having identified Pr, Co, Da and Fl, we identify the set of significant transactions: dr = D(3,9,f), dw = D(1,9,f), frc = D(3,7,t) and so forth as

defined previously. The set of significant paths, M, contains the bridges connecting each of the producer, consumer, data and flag agents. Since the paths overlap in the middle two bridges, the middle path is split into a seperate path and there are a total of five paths in this example. Each of the significant paths are circled and numbered in figure 2. The abstractin function applies ψ_M to each of the bridges and agents in the network to obtain the abstract network.

$$\alpha_I(\psi_M, \tau_M, N, M_0) = \psi_M(q_1, \tau_M, \psi_M(q_2, \tau_M, \psi_M(q_3, \tau_M, \ldots \psi_M(q_9, \tau_M, M_0))))$$

We expand the applications of ψ_M as follows.

1. Expand the application of ψ_M to q_9.

   $$\psi_M(q_9, \tau_M, M_0) = M_0 \text{ with insert } (q_9, 5, \tau_M)$$

   since q_9 is a member of path 5 as indicated in figure 2. The application of insert to q_9 results in the following

   $$\text{insert}(q_9, 5, \tau_M) = \tau_M(P(3, 9, f), M)$$

   because path 5 contains no entries at this point. Because transaction $P(3, 9, f)$ is not a significant transaction, $\tau_M(P(3, 9, f))$ is ε and no new significant transactions are added to path 5. Finaly, $\psi_M(q_9, \tau_M, M_0)$ evaluates to M_0 because all of the transactions in q_9 are insignificant.

2. The abstract network returned by $\psi_M(q_9, \tau_M, M_0)$ is passed as the abstract network to $\psi_M(q_8, \tau_M, \psi_M(q_9, \tau_M, M_0))$ which again evaluates to M_0 because q_8 is not on a significant path.

3. $\psi_M(q_7, \tau_M, \psi_M(q_8, \tau_M, M_0))$ also evaluates to M_0. Although q_7 is a member of path 4, none of the transactions in q_7 are significant transactions. Next, $\psi_M(q_6, \tau_M, \psi_M(q_7, \tau_M, M_0))$ adds two significant transactions to path 2. This is because q_6 is on path 2 and two the transactions in q_6 are significant transactions: P(1,9) is the posted write from the producer to data and D(3,7,t) is a committed delayed transaction from the consumer to data. We now have:

   $$\psi_M(q_6, \tau_M, \psi_M(q_7, \tau_M, M_0)) = 1:, 1':, 2: frc, dw, 2':, 3:, 3':, 4:, 4':, 5:, 5':$$

   which we will abbreviate M_1.

4. $\psi_M(q_4, \tau_M, M_1)$ returns M_1 unchanged. Although q_4 is on path 2, and the transaction $D(3, 7, t)$ in q_5 is a significant transaction, it is not the newest committed delayed transaction from agent 3 to agent 7 in path 2. The newest committed delayed transaction from agent 3 to agent 7 in path 2 is contained in q_6 and has already been added to the abstract network.

5. $\psi_M(q_3, \tau_M, M_1)$ adds two significant transactions to path 3 in abstract network M_1. Queue q_3 is on path 3 and two of the transactions are significant transactions. The significant transaction $D(3, 7, t)$ is kept because it is the newest committed copy of $D(3, 7, t)$ on path 3, even though younger copies exist in path 2. The significant

transaction $D(3, 9, f)$ is kept because all uncommited delayed transactions are carried over into the abstract network. After evaluating $\psi_M(q_3, \tau_M, M_1)$ we have to following abstract network, which we abbreviate M_2:

$$M_2 = 1:, 1': \ 2: frc, fw, dw\ 2': \ 3: dr, frc3': 4:, 4': \ 5: fr, 5':$$

6. ψ_M is applied to q_2 and q_1—neither of which add any transactions to M_2.

After expanding the application of α_M, we we have that the abstraction of network N from figure 1 is:

$$\alpha_I(\psi_M, \tau_M, N, M_0) = 1:, 1': \ 2: frc, fw, dw\ 2': \ 3: dr, frc3': 4:, 4': \ 5: fr, 5':$$

3 State Transition Rules

Having defined the external and internal states of PCI and PCI_A as well as an abstraction function which maps internal states to external states, we now turn our attention to defining the transition rules for concrete and abstract states. The abstraction function was carefully defined to create a structural and data abstraction that reduces arbitrary concrete instances of the producer/consumer in PCI networks to one of three abstract networks. Moreover, each abstract network contains a small number of transactions thus reducing the number of states in each abstract network.

However, the data and structural abstraction has complicated the problem of defining a set of abstract transitions such that every transition in the concrete network is modeled by some transition in the abstract network. In this section, we first give a set of rules that define the pre and post conditions for concrete PCI networks. Next we give the rules for abstract networks. The rules for concrete PCI networks will be labeled with numbers while the rules for abstract PCI_A networks will be labeled with letters. In section 4 we show that every transition in a concrete network has a corresponding transition in an abstract network.

3.1 State Transitions for PCI

The state transition rules for the internal state of PCI are the same as in [MHJG00]. The state transition rules,δ_I for the internal states are the same as in [MHJG00].

The transition rules for PCI are given in a rule-based notation. In each rule, the preconditions appear on the top and the postconditions appear on the bottom. Each preconditions uses at most the contents of two bridges which each postcondition updates at most the contents of two bridges. The rules presented here are based on a PVS model of PCI used in [MHJG00] which was in turn adopted from the original PCI specification [PCI95] and Corella's PCI model [CSZ97].

Before defining the rules, we define a set of auxilary functions:

- address$(m) = n$ returns the address for a given agent.
- next$(q, t) = q'$ returns the next queue for transaction t in queue q. The *next* function uses the static routing table to determine the next bridge for a transaction in queue q.

- $\text{opp}(q) = q'$ returns the opposite queue of queue q. The opposite queue is the queue in the same bridge but which flows in the opposite directions. Note that for all queues, $\text{opp}(\text{opp}(q)) = q$.
- $\text{append}(q, t) = (t, q_n \ldots q_0)$ creates a new queue by appending transaction t on the end of queue q.

The first two rules create new non-significant posted or delayed transactions in a master agent m.

Rule 1 (d-begin) $$\frac{\mathcal{R}(N_n) \wedge (m \neq Co) \wedge (t \neq Da) \wedge (t \neq Fl)}{\mathcal{R}(N_n[m \leftarrow append(m, D(address(m), address(t), f))])}$$

Rule 2 (p-begin) $$\frac{\mathcal{R}(N_n) \wedge (m \neq Pr) \wedge (t \neq Da) \wedge (t \neq Fl)}{\mathcal{R}(N_n[m \leftarrow append(m, P(address(m), address(t), f))])}$$

The preconditions check that the new transaction is not a significant transaction. The postcondition appends the new transaction to rear of the outgoing queue in m. Even the the abstraction discards insignificant transactions, in significant transactions are allowed for the sake of completeness with respect to the PCI definition. The next rules create new significant transactions.

Rule 3 (sig-d-begin) $$\frac{\mathcal{R}(N_n) \wedge s \notin S \wedge (s = D(Co, Fl) \vee (s = D(Co, Da) \wedge D(Co, Fl, f) \in S))}{\mathcal{R}(N_n[C \leftarrow append(C, s), S \leftarrow append(S, s)])}$$

The first of these rules issues a new significant delayed transaction. A delayed read to the flag can be issued if it does not already appear in the set of issued significant transactions S. A delayed read to the data can be issued only if it does not already appear and the delayed flag read has already been issued. The extra condition on the issuance of a delayed data read ensures that precondition two of the producer/consumer definition is met. Rule sig-d-begin does not create flag or data write transactions because we have assumed that only the flag and data read can be delayed transactions. Flag and data writes are created by the next rule.

Rule 4 (sig-p-begin) $$\frac{\mathcal{R}(N_n) \wedge s \notin S \wedge (s = P(P, D) \vee (s = P(P, F) \wedge P(P, D) \in S))}{\mathcal{R}(N_n[C \leftarrow append(C, s), S \leftarrow append(S, s)])}$$

Once again the extra precondition on the creation of a posted flag write is intended to ensure that precodition one of the producer/consumer definition is met.

The next two rules reorder or delete PCI transactions in a queue. The reordering rules depend only on adjacent transactions so we consider only pairs of transactions in the pre and post condition.

Rule 5 (swap) $$\frac{\mathcal{R}(N_n) \wedge \langle t_1, t_2 \rangle \in m}{\mathcal{R}(N_n[m \leftarrow [\langle t_1, t_2 \rangle, \langle t_2, t_1 \rangle]])}$$

Where t_1 and t_2 are one of the following pairs of transactions: like build a table of t_1 and t_2 as in the specificaiton, but repl all yes/no with yes and claim non-det to say that this is the most general model and other yes/no choices are executed.

The deletion rule for completions is slightly modified from those given in the PCI specification.

Rule 6 (r-discard) $\dfrac{\mathcal{R}(N_n) \wedge t \in q \wedge t \in (D(a,b,f), C(a,b,f))}{\mathcal{R}(N_n[q \leftarrow q[x \leftarrow \varepsilon]])}$

The specification allows a completion to be dropped only when it has been in the same bridge for more than 2^{15} clock cycles. We relax that restriction to allow a completion to be dropped at any time.

The next set of rules cover the movement of delayed transactions through bridges and agents. The first two rules model the first attempt of a transaction on a bus. After a delayted transaction is attempted on the bus it becomes committed whether or not it is actualy latched in the next bridge or agent. For example, the transaction D(3,8,f) in the bottom queue of bridge six can be attempted on the bus between bridges six and four. Bridge four may or may not accept D(3,8,f), in either case, D(3,8,f) is marked as *committed* indicating that is has been attempted on the bus. The first rule models the case in which the transaction is not accepted by the next bridge or agent.

Rule 7 (d-commit) $\dfrac{\mathcal{R}(N_n) \wedge q = \{t_1 \ldots t_n, D(a,b,f)\rangle}{\mathcal{R}(N_n[q \leftarrow \{t_1 \ldots t_n, D(a,b,t)\rangle])}$

The precondition checks that uncommitted form of a delayed transaction, $D(a,b,f)$, is at the front of the queue. The post condition replaces the uncommitted delayed transaction with a committed delayed transaction, $D(a,b,t)$. Now that the transaction is committed, it can not be deleted. The next rule models the case in which the transaction is attempted and accepted by the next bridge.

Rule 8 (d-commit2) $\dfrac{\mathcal{R}_\alpha(M_n) \wedge q = \{t_1 \ldots t_n, D(a,b,f)\rangle}{\begin{array}{l}\mathcal{R}(N_n[\ q \leftarrow q[D(a,b,f) \leftarrow D(a,b,t)],\\ \qquad next(q, D(a,b,f)) \leftarrow append(next(q, D(a,b,f)), D(a,b,f))])\end{array}}$

——————— The precondition is the same, but the postcondition places a new uncommitted copy of the transction in the queue in the next bridge. The queue that attempted the transaction, q, keeps its copy of the committed transaction for matching with the completion later.

The next rule models the latching of an already committed transaction to the bridge. After a transaction has been committed, the local bridge will reattempt the transaction on the bus whenever the next bridge or agent does not contain a copy of the transaction in outgoing queue of the next bridge or a matching completion in the incoming queue of the next bridge. For example, the transaction D(3,7,t) in bridge six of figure 1 could latch to agent seven because D(3,7,t), D(3,7,f) nor C(3,7,t) appear in agent seven but D(3,7,t) in bridge three can not latch to bridge seven because D(3,7,t) aleary appears in bridge seven.

Rule 9 (d-latch) $\dfrac{\begin{array}{c}\mathcal{R}(N_n) \wedge q = \{t_1 \ldots t_n, D(a,b,t)\rangle \wedge\\ D(a,b,t) \notin next(q, D(a,b,t)) \wedge\\ D(a,b,f) \notin next(q, D(a,b,t)) \wedge\\ C(a,b,f) \notin opp(next(q, D(a,b,t))) \wedge\\ C(a,b,f) \notin oppq\end{array}}{\mathcal{R}_\alpha(M_n[next(q, D(a,b,t)) \leftarrow append(next(q, D(a,b,f), D(a,b,f)))])}$

The preconditions check that the transaction nor its completion are found in the next bridge. The postcondition places a new uncommitted copy of the transaction in the next bridge.

Eventualy the transaction reaches its target. When a delayed transaction reaches its target, the target agent creates a completion transaction and deletes delayed transaction. In figure 1, the delayed transaction from agent five to agent two has just ended at agent two. Note the completion, C(5,2), in the outgoing queue of agent two and the trail of delayed transactions, D(5,2,t), leading back to agent 5.

Rule 10 (d-end)
$$\frac{\mathcal{R}(N_n) \wedge q = \{t_1 \ldots t_n, D(a,b,t)\rangle \wedge addressq = b \wedge C(a,b,f) \notin opp(q)}{\begin{array}{l}\mathcal{R}(N_n[\ opp(q) \leftarrow append(opp(q), C(a,b,f)),\\ \qquad C \leftarrow append(C, (D(a,b,t), q)),\\ \qquad q \leftarrow q[D(a,b,t) \leftarrow \varepsilon]])\end{array}}$$

The precondition checks that the transaction has indeed reached the target agent by comparing the address of the agent with the destination address of $D(a,b,t)$. If the address matches and the completion has not already been issued, then the postcondition appends the completion on the outgoing queue, adds $D(a,b,t)$ to the set of completed transactions C and deletes its copy of $D(a,b,t)$. Note that we have relaxed the use of local master IDs by using the ID of the issuing agent rather than the ID of the local bridge in the completion.

The completion then moves from bridge to bridge following the trail of committed delayed transactions back to the originating master.

Rule 11 (d-comp)
$$\frac{\mathcal{R}(N_n) \wedge q = \{t_1 \ldots t_n, D(a,b,t)\rangle \wedge C(a,b,f) \in opp(next(q, D(a,b,t)))}{\begin{array}{l}\mathcal{R}(N_n[\ opp(next(q, D(a,b,t))) \leftarrow opp(next(q, D(a,b,t)))[C(a,b,f) \leftarrow \varepsilon],\\ \qquad q \leftarrow q[D(a,b,t) \leftarrow \varepsilon],\\ \qquad opp(q) \leftarrow append(oppq, C(a,b,f))])\end{array}}$$

The precondition checks that the matching completion appears in the opposite queue of the next bridge. The postcondition copies the completion into the opposite queue of the bridge, deletes the committed delayted transaction and deletes the completion from the opposite queue of the next bridge. If the queue containing $D(a,b,t)$ is the issuing master agent, then the rule leaves a copy of the completion in the incoming queue of the agent but deletes the transction from the outgoing queue. We use the list of completions in the incoming queue for an agent as auxilary state to record the order in which transactions were completed at a master agent.

The rule presented here differs from the original PCI specification in that completions cary the addresses of the delayed transaction they are completing. This is done to prevent completion stealing [Cor96] which can be exploited to violate the producer/consumer property as explained in section 2.1. We also differ slightly from the rules presented in [?] in that completions carry the master ID of the orginating master rather than the ID of the next local bus. For example, in figure 1, the completion in agent two should contain local master ID four, C(2,4,t), rather than five, C(2,5,t). We use originating master IDs rather than local master IDs because it simplifies the rules and does not effect transaction ordering.

The next two rules governs the movement of posted transactions. The movement of posted transactions is much simpler because posted transactions simply move from bridge to bridge until reaching their destination. Posted transactions do not leave trails of committed copies as delayed transactions do. The first rule governs the movement of a posted transaction.

Rule 12 (p-complete)
$$\frac{\mathcal{R}(N_n) \wedge (q = \{t_1 \ldots t_n, P(a,b,f)))}{\mathcal{R}(N_n[\ q \leftarrow q[P(a,b,f) \leftarrow \varepsilon],\ next(q,P(a,b,f)) \leftarrow append(next(q,P(a,b,f)),P(a,b,f))])}$$

The precondition checks that a posted transaction appears at the head of the queue and the precondition moves the transaction from the queue to the next. The final posted transaction rule completes a posted transaction that has reached the head of its target queue.

Rule 13 (p-end)
$$\frac{\mathcal{R}(N_n) \wedge (q = \{t_1 \ldots t_n, P(a, address(q), f)\rangle)}{\mathcal{R}(N_n[\ q \leftarrow \{t_1 \ldots t_n\rangle,\ C \leftarrow append(C,(P(a,address(q),f),q))])}$$

The precondition checks that a posted transaction has reached the head of the queue to which it was addressed. The postcondition removes the transaction from q and appends it to C.

A PCI producer/consumer trace, denoted σ, is finite set of external and internal state pairs, $(\alpha_I(\psi_M, \tau_M, N_i, M_0), N_i)$, begining with the initial state, N_0, related by transitions from δ_I. We say that the the *ith* state in a trace, $(\alpha_I(N_i), N_i)$, is equal to i applications of rules from δ_I. We write the application of rule number J to transaction t in queue q of state N_i as $\delta_J(q,t,N_i)$ so that $N_{i+1} = \delta_J(q,t,N_i)$ for any q, t or rule J, and we have

$$\sigma = (\alpha_I(\psi_M, \tau_M, N_0, M_0), N_0),\ (\alpha_I(\psi_M, \tau_M, N_1, M_0), N_1),\ \ldots (\alpha_I(\psi_M, \tau_M, N_n, M_0), N_n)$$

A PCI producer/consumer trace includes four pre-identified producer/consumer agents which act as the producer, consumer, data and flag agents during the execution and a set of pre-identified read and write transactions that indicate the flag read and write and data read and write transactions. The pre-identified agents are used to create the significant path set M which is used by the abstraction function.

A PCI producer/consumer trace ends when the consumer data read completes at the data agent. This might be a good place to make the arguement for completion. If we assume that each of the significant transactions always eventualy get to the front of a bridge, then we can assume that the trace will eventualy terminate using the liveness results of [CSZ97].

3.2 State Transitions for $\mathbf{PCI}_A$

The state transition rules, δ_a, for PCI_A inductively define the set of reachable PCI_A states characterized by a reachability predicate $\mathcal{R}_\alpha$. A state M_n is reachable by the application of rules starting from M_0 if and only if $\mathcal{R}_\alpha(M_n)$. The general form of each rule is:

$$\frac{\mathcal{R}_\alpha(M_n) \wedge P(M_n)}{\mathcal{R}_\alpha(M_n[\text{substitution list}])}$$

Where $P(M_n)$ is a predicate on M_n that defines the preconditions for the rule and the new reachable state is created by sequentialy performing the substitions in the substition list. In the rules that follow we abreviate $p.M_n$ (or, path p in configuration M_n) with p. We use similar abbreviations for transactions.

The state transtion rules use several auxillary functions defined below:

- next$(p, t) = p'$ denotes the next path for transaction t in path p.
- prev$(p, t) = p'$ denotes the previous path for a transaction t in path p.
- opp$(p) = p'$ denotes the opposite path of path p. The opposite path is the path of queues in the opposite direction. Note that for all paths, opp(opp$(p)) = p$.
- compl$(t) = t'$ denotes the completion, t', for transaction t. For example, compl$(fwc) = cfw$.
- interleave$(p, t) = p'$ denotes the path, p', created by moving transaction t to a non-deterministicaly chosen location in the prefix of t in p. For example, interleave$(\{a, b, c, t, d\}, t)$ is equal to $\{t, a, b, c, d\rangle$, $\{a, t, b, c, d\rangle$, $\{a, b, t, c, d\rangle$ or $\{a, b, c, t, d\rangle$ but not $\{a, b, c, d, t\rangle$.

Given the auxillary functions, δ_a is defined inductively by the following set of rules. The rules were created after the structural and data abstractions were defined. The rules were designed so that the concrete PCI model would be a refinement of this abstract model. Specificaly, we ensure that the abstraction of a concrete state that satisfies the precondition to some concrete rule also satisfies the precondition of at least one abstract rule such that the abstraction of the concrete postcondition is equal to the abstract postcondition. The preconditions of each rule refer to at most two paths in the abstract network and change at most three paths at a time in the network. At any given time, one rule may apply to several paths in the same state.

The first two rules model transaction deletion and reordering in the abstract network.

Rule A (r-discard) $\dfrac{\mathcal{R}_a(M_n) \wedge x \in p \wedge x \in \{dr, fr, cdr, cfr\}}{\mathcal{R}_a(M_n[p \leftarrow p[x \leftarrow \varepsilon]])}$

Rule B (swap) $\dfrac{\mathcal{R}_a(M_n) \wedge \{a, b\rangle \in p}{\mathcal{R}_a(M_n[p \leftarrow p[\{a, b\rangle \leftarrow \{b, a\rangle]])}$

Where (a, b) is selected from one of the following pairs:

$$dw, fw \text{ can pass } dr, fr, drc, frc, cdr, cfr$$
$$dr, fr, drc, frc, cdr, cfr \text{ can pass } dr, fr, drc, frc, cdr, cfr$$

The preconditions to either rule depend on the presence of one or two transactions in a path. The postconditions either discard or reorder the transactions to mirror the effects of reordering or deleting significant transactions in PCI.

The next set of rules model the behavior of significant delayed transactions in PCI_A. First, the d-begin rule provides a mechanism for adding D transactions to the abstract network.

Rule C (d-begin) $\dfrac{\mathcal{R}_a(M_n) \wedge s \notin S_\alpha \wedge (p = Co) \wedge (s = fr \vee (s = dr \wedge fr \in S_\alpha))}{\mathcal{R}_a(M_n[p \leftarrow append(p, s_f), S_\alpha \leftarrow append(S_\alpha, s_f)])}$

We check if a transaction has been issued by checking the set of issued transactions, S_α. We also check that the flag read has been issued before the data read can be issued. If the transaction has not been issued, then it is issued by appending an uncommitted copy of s to p. Through the remainder of this paper, significant transactions will be denoted s, with uncommitted copies denoted s_f (such as fr or dr) and committed copies denoted s_t(such as frc or drc).

The next three rules model attempting to latch a delayed transaction to the next queue. The first rule models the case where s is attempted, but not latched to the next queue.

Rule D (d-commit) $$\frac{\mathcal{R}_\alpha(M_n) \wedge s_f \in p}{\mathcal{R}_\alpha(M_n[p \leftarrow p[s_t \leftarrow \varepsilon, s_f \leftarrow s_t]])}$$

The only precondition is that s_f appears in p. The postcondition deletes all previous copies of s_t by replacing them with ε and replaces s_f with s_t. The next two rules commit and latch a transaction to the next queue. There are two cases to consider. In the first case, the next queue is on the same path.

Rule E (d-commit2) $$\frac{\mathcal{R}_\alpha(M_n) \wedge s_f \in p}{\mathcal{R}_\alpha(M_n[p \leftarrow p[s_t \leftarrow \varepsilon, s_f \leftarrow \{s_t, s_f\rangle]])}$$

The only precondition is that s_f is present in p. The postcondition replaces s_t with ε but s_f is replaced by s_f,s_t to model the new copy of s_f in the next queue. The final d-commit rule models the case where s_f is both committed and latched to the next queue and the next queue lies on the next path.

Rule F (d-commit3) $$\frac{\mathcal{R}_\alpha(M_n) \wedge p = \{t_1, \ldots, t_n, s_f\rangle}{\begin{array}{l}\mathcal{R}_\alpha(M_n[\ p \leftarrow p[s_t \leftarrow \varepsilon, s_f \leftarrow s_t],\\ \qquad next(p,s) \leftarrow append(next(p,s), s_f)])\end{array}}$$

The precondition checks that s_f lies at the head of p. The postcondition is the same as d-commit2, except that s_f is replaced by only s_t and the new copy of s_f is appened to next(p, s).

The next two rules latch a previously committed transaction to the next queue. The next queue may lie on either the same abstract path or the next abstract path. Note that all states that satisfy d-latch-path precondition also satisfy d-latch precondition. This is because a path may or may not have more queues in front of the queue containing t_t, and since we ignore queue boundaries, we must allow either possibility. The first rule models the case where the next queue lies on the same path.

Rule G (d-latch) $$\frac{\begin{array}{c}\mathcal{R}_\alpha(M_n) \wedge s_t \in p \wedge s \notin next(p)\\ \wedge compl(s) \notin opp(p)\\ \wedge compl(s) \notin opp(next(p))\end{array}}{\mathcal{R}_\alpha(M_n[p \leftarrow p[s_t \leftarrow \{s_t, s_f\rangle]])}$$

The preconditions check that the transaction has not already latched into the next path and that there is not a pending completion in the opposite or opposite next path. The postcondition replaces s_t with s_t,s_f to model the placement of s_f in the next queue. The second rule models the case where the next queue lies in the next path.

Rule H (d-latch-path)
$$\frac{\begin{array}{c}\mathcal{R}_\alpha(M_n) \wedge p = \langle t_0 \ldots t_n, s_t\rangle \wedge s \notin next(p,s) \\ \wedge compl(s) \notin opp(p) \\ \wedge compl(s) \notin opp(next(p,s))\end{array}}{\mathcal{R}_\alpha(M_n[next(p,s) \leftarrow append(next(p,s), s_f)])}$$

The preconditions check that s_t has reached the head of the path and that neither s nor it's completion can be found in p, next(p,s) or their opposite paths. The postcondition simply appends s_f to the next path.

The next three rules model three ways to end a delayed transaction in the abstract state space. The preconditions to each rule are the same. The preconditions check that the transaction has reached the head of the final path and that the completion has not already been created in the opposite path. Each of the postconditions append s_t to the set of completed transactions, C_α. Once again, information lost in the structural abstraction requires defining several rules. Three rules are needed because deleting the final s_t transaction results in a new state in which the next-newest s_t transaction appears somewhere in either the final path or the previous path. The first rule models the case in which path p contains more than one queue and s_t is not the only transaction in the queue. In this case, the next-newest copy of s_t can appear between any two transactions in p.

Rule I (d-end)
$$\frac{\mathcal{R}_\alpha(M_n) \wedge (p = \langle t_0, \ldots, t_n, s_t\rangle) \wedge (compl(s) \notin opp(p)) \wedge (s = (D, \_, p, t))}{\begin{array}{l}\mathcal{R}_\alpha(M_n[\ opp(p) \leftarrow append(opp(p), compl(s)), \\ \qquad C_\alpha \leftarrow append(C_\alpha, (s_t, p)), \\ \qquad p \leftarrow interleave(p, s_t)])\end{array}}$$

The final clause in the postcondition is a disjunction of possibilities that allows the next-newest copy of s_t to appear between any two transactions in p. The actual next state created by an application of rule d-end is a non-deterministic choice of any of the possibilities generated by this rule. The second rule models the cases in which the s_t is the only transaction in the final queue on path p and the next-newest transaction is at the head of the previous queue—also contained in path p. In this case, the next-newest copy of s_t remains at the head of path p.

Rule J (d-end2)
$$\frac{\mathcal{R}_\alpha(M_n) \wedge (p = \langle t_0, \ldots, t_n, s_t\rangle) \wedge (compl(s) \notin opp(p)) \wedge (s = (D, \_, p, t))}{\mathcal{R}_\alpha(M_n[opp(p) := (opp(p), compl(s)), C_\alpha := append(C_\alpha, (s_t, p))])}$$

The postcondition to rule d-end2 appends s_t to the completed transaction list, creates the completion in the opposite path and leaves path p unchanged. The third rule models the case in which path p contains only one queue. In this case, deleting s_t deletes the only copy of s_t in the queues of path p so that the next newest copy of s_t is the one preserved in the previous path.

Rule K (d-end3)
$$\frac{\mathcal{R}_\alpha(M_n) \wedge (p = \langle t_0, \ldots, t_n, s_t\rangle) \wedge (compl(s) \notin opp(p)) \wedge (s = (D, \_, p, t))}{\begin{array}{l}\mathcal{R}_\alpha(M_n[\ opp(p) \leftarrow append(opp(p), compl(s)), \\ \qquad C_\alpha \leftarrow append(C_\alpha, (s_t, p)), \\ \qquad p \leftarrow \langle t_0 \ldots t_n\rangle])\end{array}}$$

The postcondition to rule d-end3 is the same as the postcondition to rule d-end2 except the only copy of s_t in path p is deleted.

The final three rules for delayed transactions match committed delayed transactions with their completions. The first two rules model the cases in which both the delayed transaction and its completion appear in the same path. The preconditions for both rules check that that s_t appears in p and that compl(t) appears in path opp(p). In both cases the completion is unchanged because it is moved from one queue to the next in path opp(p), which does not change its location in opp(p). The first rule models the case in which s_t is not the only copy of s_t in path p.

$$\textbf{Rule L (d-comp)}\ \frac{\mathcal{R}_\alpha(M_n) \wedge (p = \langle t_0 \ldots t_n, s_t, t_{n+1} \ldots t_m\rangle) \wedge (compl(s) \in opp(p))}{\mathcal{R}_\alpha(M_n[\ p \leftarrow interleavep, s_t \vee \\ p \leftarrow \langle s_t, t_0 \ldots t_m\rangle])}$$

If s_t is not the only copy of s_t in p, then deleting s_t allows the next-newest copy of s_t to appear in path p. The next-newest copy of s_t can appear anywhere in path p before the location of the previous newest copy of s_t. As in d-end, the disjunction in the post-conditin allows non-deterministic choice of the next state. The second rule models the case in which s_t is in the last queue in path p and is therefore the last copy of s_t in p.

$$\textbf{Rule M (d-comp2)}\ \frac{\mathcal{R}_\alpha(M_n) \wedge (p = \langle t_0 \ldots t_n, s_t, t_{n+1} \ldots t_m\rangle) \wedge (compl(s) \in opp(p))}{\mathcal{R}_\alpha(M_n[p \leftarrow \langle t_0 \ldots t_m\rangle])}$$

In this case, the postcondition simply deletes the last copy of s_t and the next-newest copy of s_t is either already present in the previous path, or p was the master path for s and compl(s) is kept in opp(p) so that s will not be reissued.

The final two completion rules model the cases in which the transaction and its completion are in queues on different paths. The preconditions to both rules check that s_t and compl(s) have reached the heads of their respective paths. Both rules include postconditions that delete compl(s) from opp(next(p, s)) and append a new copy of compl(s) to opp(p). The first rule models the case in which other copies of s_t appear in other queues in path p.

$$\textbf{Rule N (d-comp-next)}\ \frac{\mathcal{R}_\alpha(M_n) \wedge (p = (t_0 \ldots t_n, s_t)) \wedge (s_t \notin next(p,s)) \wedge \\ (opp(next(p,s)) = (compl(s_t), t'_0 \ldots t'_n))}{\mathcal{R}_\alpha(M_n[\ p \leftarrow interleave(p, sst), \\ opp(p) \leftarrow append(opp(p), compl(s)), \\ opp(next(p,s)) \leftarrow opp(next(p,s))[compl(s) \leftarrow \varepsilon]])}$$

Deleting s_t from p allows the next-newest copy of s_t to appear somewhere in path p. All possible locations of s_t are considered. The second rule models the case in which s_t is in the only queue in path p.

$$\textbf{Rule O (d-comp-next2)}\ \frac{\mathcal{R}_\alpha(M_n) \wedge (p = \langle t_0 \ldots t_n, s_t\rangle) \wedge (s_t \notin next(p,s)) \wedge \\ (opp(next(p,s)) = \langle compl(s_t), t'_0 \ldots t'_n\rangle)}{\mathcal{R}_\alpha(M_n[\ opp(p) \leftarrow append(opp(p), compl(s)), \\ opp(next(p,s)) \leftarrow opp(next(p,s))[compl(s) \leftarrow \varepsilon]])}$$

In this case, deleting s_t from path p does not allow a next-newest copy of s_t to appear in p.

We now define three transition rules for posted transactions—which are comparatively simple. Posted transactions are issued at their master agent and simply move through the paths until they reach the target agent. The first rule allows posted entries to be created if they have not already been created.

Rule P (p-begin) $\dfrac{\mathcal{R}_\alpha(M_n) \wedge s \notin S_\alpha \wedge (p = Pr) \wedge (s = dw \vee (s = fw \wedge dw \in S_\alpha))}{\mathcal{R}_\alpha(M_n[p := append(p,s), S_\alpha := append(S_\alpha, s)])}$

The predonondition checks that the transaction has not already been issued, that path p is the Producer path and requires that dw is issued before fw.

The precondition checks the abstract issued significant transaction set, S_α, to ensure that s hasn't already been issued. If s has not already been issued, the postcondition appends s to p and adds s to S_α.

The next rule for posted transactions models the movement of a posted transaction from one path to the next.

Rule Q (p-complete) $\dfrac{\mathcal{R}_\alpha(M_n) \wedge (p = \{t_0 \ldots t_n, s\rangle)}{\mathcal{R}_\alpha(M_n[p \leftarrow \{t_0 \ldots t_n\rangle, next(p) \leftarrow append(next(p), s)])}$

The precondition checks that posted transaction s has reached the head of path p. The postcondition deletes s from path p and appends it to next(p).

The final posted transaction rule ends a posted transaction that has reached the head of its destination path.

Rule R (p-end) $\dfrac{\mathcal{R}_\alpha(M_n) \wedge p = \{t_0 \ldots t_n, s\rangle \wedge s = (P, \_, p)}{\mathcal{R}_\alpha(M_n[p \leftarrow \{t_0 \ldots t_n\rangle, C_\alpha \leftarrow append(C_\alpha, s)])}$

The precondition checks that s has reached the head of path p and that s is addressed to path p. The postcondition removes s from $p$and appends it to C_α.

The last rule, noop, introduces a convinient way to model stuttering through repeated noops in the abstract state.

Rule S (noop) $\dfrac{\mathcal{R}_\alpha(M_n)}{\mathcal{R}_\alpha(M_n[\,])}$

A producer/consumer trace from PCI_A is a finite set of external states related by applications of rules from δ_a. We say that the the ith state in a trace, M_i, is equal to i applications of rules from δ_a. We write the application of rule E to transaction s in path p of state M_i as $\delta_E(p, s, M_i)$ so that $M_{i+1} = \delta_E(p, s, M_i)$ for any p, s or rule E. A producer/consumer trace of PCI_A, σ_A, is a finite sequence of M_is starting with the initial abstract state M_0.

$$\sigma_A = M_0, \ldots M_n$$

Abstract state M_0 is the abstract state in which all paths in M are empty. Abstract state M_n is the abstract state in which the Consumer Data read transaction has completed at the data agent.

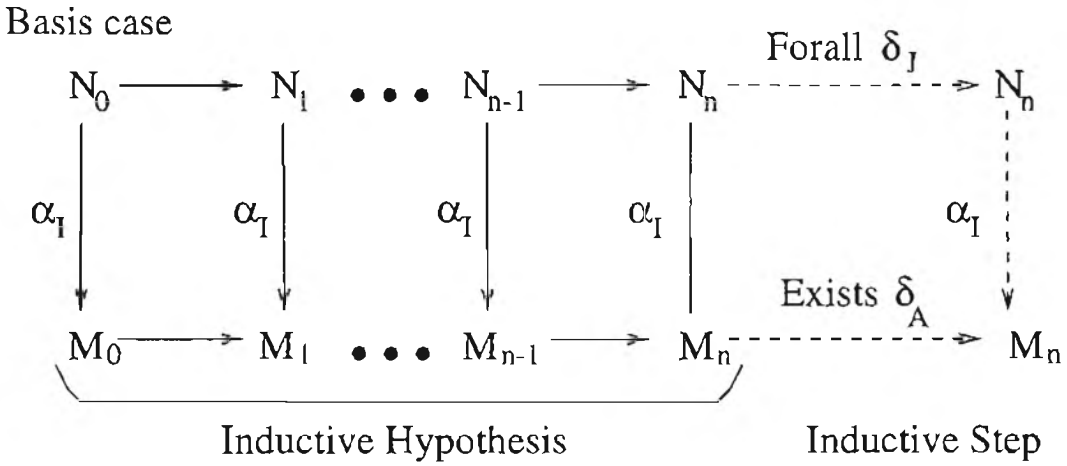

Figure 3: Outline of proof that PCI is a refinement of PCI_A.

4 PCI is a Refinement of PCI_A

We need to show that the following relationship holds:

$$\forall \sigma \in \text{PCI}. \quad \{\sigma = (M_0, N_0), (M_1, N_1), (M_2, N_2) \ldots (M_n, N_n) \\ \Rightarrow \exists \sigma_A \in \text{PCI}_A . \sigma_A = M_0, M_1, M_2 \ldots M_n\}$$

where each M_i in σ is created by applying the abstraction mapping to concrete state N_i for a particular instance of the producer/consumer property. The relationship will be shown by induction on the length of σ. In the concrete trace, we will assume *fairness* for significant transactions. More specificaly, we assume that each significant transaction is always eventualy affected by the post-condition of a rule. This assumption avoids infinite noop transitions in the abstract state space and is justified by the liveness results from [CSZ97] which clarify the conditions under which a PCI transaction may be assumed to always eventualy complete.

The proof is outlined in figure 3. The concrete trace is shown on the top of the figure while the abstract trace is shown at the bottom. The abstract and concrete traces are related by the abstraction mapping, α_M. The basis case is shown on the left and requires us to show that the abstraction of the initial concrete state is equal to the abstract initial state. The inductive hypothesis assumes that for any trace of length n, there exists some abstract trace of length n such that the abstract and concrete states are related by the abstraction mapping. The inductive step requires is to show that for every application of every concrete rule, there exists an application of an abstract rule such that the abstraction of the next concrete state equals the next abstract state. We now give the inductive proof the PCI is a refinement of PCI_A in more detail.

4.1 Basis Case

We show that the abstraction of the initial concrete state, $M' = \alpha_I(\psi_M, \tau_M, N_0, M_0)$, is equal to the abstract initial state, M_0. Recall that neither N_0 nor M_0 contain any transactions. Since α_I only removes transactions from N_0, we conclude that the abstraction of N_0 does not contain any transactions either. The paths in M' and M_0 are the same for a given instance of producer/consumer, and neither contain any transactions, so we have that $M_0 = \alpha_I(\psi_M, \tau_M, N_0, M_0)$.

4.2 Inductive Step

Assume that for any n,

$$\begin{aligned}\forall \sigma_n \in \text{PCI}. \quad & \{\sigma_n = (M_0, N_0), (M_1, N_1), (M_2, N_2) \ldots (M_n, N_n) \\ & \Rightarrow \exists \sigma_{An} \in \text{PCI}_A . \sigma_{An} = M_0, M_1, M_2 \ldots M_n\}\end{aligned}$$

and show that for any concrete rule δ_J, there exists some abstract rule δ_A

$$\begin{aligned}\forall \sigma'_n \in \text{PCI}. \quad & \{\sigma'_n = (M_0, N_0), (M_1, N_1), (M_2, N_2) \ldots (M_n, N_n), (\alpha_I(\delta_J(N_n)), \delta_J(N_n)) \\ & \Rightarrow \exists \sigma'_{An} \in \text{PCI}_A . \sigma'_{An} = M_0, M_1, M_2 \ldots M_n, \delta_a(M_n)\}\end{aligned}$$

this will be done by showing that for any rule numbered J acting on any transaction t in any queue q of network state N_n, there exists an abstract rule lettered A acting on some path p and significant transaction s such that the abstraction of the next state created by δ_J on N_n is equal to the next state created by δ_A on the abstraction of N_n.

$$\begin{aligned}&\forall J, q, t, N_n. \\ &\quad \exists A, p, s. \\ &\qquad \alpha_I(\delta_J(q, t, N_n)) = \delta_A(p, s, \alpha_I(N_n))\end{aligned}$$

Note that if $\alpha_I(\delta_J(q, t, N_n)) = \alpha_I(N_n)$ we chose δ_A to be the noop rule, δ_S. This complicates matters somewhat because to show that PCI refines PCI_A in the presence of a noop rule in PCI_A, we must show that only a finite number of noop rules are applied in any PCI trace [?]. We can see that only finite applications of noop appear in a PCI_A producer/consumer trace because we assume that a significant transaction will always eventauly be affected by a rule application.

We give the sub-proofs for each concrete rule applied to any transaction in any queue. The general approach in each sub-proof is to identify the applications that chage the abstraction of the next state. For each application that does change the abstraction of the next state, we choose an abstract rule such that:

1. The abstract rule is enabled if the concrete rule is enabled. Specificaly, if the precondition to the concrete rule is satisfied, then the precondition to the abstract rule is satisfied by the abstraction of the concrete state.
2. The postcondition of the abstract rule is equal to the abstraction of the postcondition of the concrete rule.

In many cases it will be neccesary to consider several abstract rules to cover the effects of a single concrete rule.

Rule 1 d-begin. Concrete rule d-begin creates new delayed transactions, but not new *significant* delayed transactions. Hence any transactions appended to an agent by d-begin are abstracted away in the corresponding abstract state. Because d-begin never updates the abstract state, the corresponding abstract rule for d-begin is Rule S, the noop rule. Note that only a finite number of d-begin and corresponding noop transitions are allowed because we assume that a significant transaction always eventualy gets a transition.

Rule 2 p-begin. Similar to d-begin.

Rule 3 significant d-begin. Significant d-begin can create a new significant delayed transaction. The precondition to rule 3 checks that the soon-to-be created significant transaction does not already exist in the significant transaction set. The corresponding abstract rule is the abstract d-begin, rule C. The precondition to rule C checks that the significant transaction does not already appear in the abstract significant transaction set. The concrete precondition implies the abstract precondition because the significant transaction sets are identical.

The postcondition to rule 3 results in a new concrete state, N'_n, in which the new significant transaction, s, is appended to the end of the corresponding agent, a. This change modifies the abstraction of the next state to include s at the end of the path containing a because

$$\psi_M(\text{append}(s, a), \tau_M, M) = \text{insert}(\text{append}(s, a), \tau_M)$$

Insert applies τ_M to every transaction in a resulting in the same abstract state as M_n with the addition of • to the end of the path containing a. This new abstract state is the postcondition of rule C. Since the abstraction of the next state of significant-d-begin can be duplicated by rules from PCI_A, every next state created by significant-d-begin has a corresponding next state in PCI_A.

Rule 4 sig-p-begin. Similar to sig-d-begin.

Rule 5 swap. The preconditions for swap check that two interchangeable transactions are adjacent in a queue. If the transactions are interchangeable, then the transactions are swapped in the next state. This rule changes the abstraction of the next state only when both transactions are significant transactions. Suppose that t is an insignificant transaction, then

$$\text{insert}((t, s), m, \tau_M) = \text{insert}((s, t), m, \tau_M)$$

since t is replaced with ε in both cases.

If swap does involve two significant transactions, s_1 s_2, then the abstraction of the next state includes s_1 and s_2 with their order reversed. We now show that any two interchangeable significant transactions in state N_n can also be interchanged by an abstract rule in state M_n. Any posted significant transaction (*pdw* or *pfw*) can pass any other transaction except another posted transaction and any delayed transaction (*fr*,*dr*,*drc* or *frc*) or completion (*cdr* or *cfr*) can pass any other transaction except a posted transaction. These correspond to the reorderings possible under the rule swap.

Note that the abstract swap rule does allow reoderings which may not be allowed in a corresponding abstract state. For example, suppose a queue contained a flag write and

a flag read separated by an insignificant posted transaction. The flag read and flag write can not pass eachother in the concrete state because of the intervening posted transaction. However, in the abstract state the flag read and write are adjacent because the posted transaction is ignored. Without the posted transaction in the abstract state, the flag read and write can pass. We are not concerned by such spurious reorderings because the refinement relation allows the abstract model to include more behaviors than the concrete model.

Rule 6 r-discard. A r-discard transition updates the abstraction of the next state only if a significant transaction is discarded. Discarding an insignificant transaction in the concrete state does not affect the abstraction of the next state because for any insignificant transaction a, $\tau_M(a) = \tau_M() = \varepsilon$.

The precondition to the discard transition depend on the presence of an uncommited delayed transaction or a delayed transaction completion. For significant transactions, the abstraction of this precondition implies the precondition to abstract rule A which allows any significant uncommited delayed transaction, fr or dr, or significant delayed transaction completion, cdr or cfr, to be dropped at any time.

The abstraction of the next state created by dropping significant transaction s, is the same abstract state without transaction s. The same abstract state is created by the postcondition of rule A. Any new abstract state generated by concrete transation 6 can be created from the corresponding abstract state by abstract rule A.

Rule 7 d-commit. Committing a transaction on a bus does not neccesarily mean that the transaction is latched to the next bridge. Rule d-commit models the case in which the transaction does not latch to the next bridge, rule 8 models the case in which the transaction is latched to the next bridge and is handled next.

Suppose that $N'_n = \delta_7(q, t, N_n)$ for some q and t in N_n and that the preconditions to δ_7 are met be q and t in state N_n. Transaction t is either an insignificant or significant transaction. If t is not a significant transction, then t_f is replaced by t_t in queue q in state N'_n and $\alpha_M(N'_n) = \alpha_M(N_n)$ because neither t_t nor t_f appear in the abstractions of either N_n or N'_n.

If t is a significant transaction s, then q lies on a significant path p and the effect of δ_7 on $\alpha_M(N'_n)$ is mirrored by rule D on transaction s in path p: $\delta_D(p, s, \alpha_M N_n)$. We first observe that the precondition to δ_D is satisfied by $\alpha_M N_n$. Rule δ_D requires that $s_f \in p$; which is satisfied because δ_7 requires that $t_f \in q$, and we have that $q \in p$ so $\psi_M(q)$ places s_f on path p in $\alpha_M N_n$.

Next, we show that $\alpha_M \delta_7(q, t, N_n) = \delta_D(p, s, \alpha_M N_n)$. Rule δ_7 on N_n replaces t_f with t_t in q. It is a property of PCI networks that $t_t \in q'$ for all queues q' between q and the source of t [JHM99]. Suppose that q' is the queue behind q on the path from the source of t to q and that q' lies on path p'. There are now two cases to consider, (1) $p = p'$ and (2) $p \neq p'$.

First, let $p = p'$, then $\psi_M(q, q', N_n)$ places s_t behind s_f on path p in $\alpha_M N'_n$. However, if we replace the s_f in q with s_t then the s_t in q' is no longer the newest copy of s_t in path p so $\psi_M(q, q', \delta_7 N_n)$ no longer contains the copy of s_t in q' and the s_f in q is replaced by s_t so that only one copy of s_t now appears on path p. This effect is captured by δ_D by replacing s_t in p with ε and replacing s_f with s_t.

Second, suppose $p \neq p'$. In this case, there are no copies of s_t in path p because q is

the last queue on path p (otherwise, $p = p'$). Replacing s_f with s_t in q has no effect on $\psi_M(q, \delta_7 N_n)$ other than to replace s_f with s_t in p. This is because s_t in q is now the newest copy of s_t in p since the previous newest copy of s_t was in q' which lies on a different path p'. We again use δ_D to model this effect since replacing s_t in p has no effect on p because $s_t \notin p$ and replacing s_f with s_t is exactly the effect of δ_7.

Rule 8 d-commit2. Rule d-commit2 models the case in which the first attempt of the transaction on the bus results in its latching to the next bridge. Suppose that $N'_n = \delta_8(q, t, N_n)$ and that (q, t, N_n) satisfy the precondition to δ_8. If t is not a significant transaction, then $\alpha_M N'_n = \alpha_M N_n$ because neither t_t nor t_f appear in $\alpha_M N'_n$. In this case, the effect of δ_8 on the abstract next state is mirrored by the noop transition δ_S.

Next, suppose that t is a significant transaction labeled s and that q lies on path p with $q' = \text{next}(q, t)$ and q' on path p'. The postcondition of transition δ_8 appends a copy of s_f to q' and replaces s_f in q with s_tThere are two cases to consider: (1) $p = p'$ and (2) $p \neq p'$.

Case (1): $p = p'$. In this case, the effects of δ_8 on $\alpha_M N'_n$ are mirrored by abstract transition δ_E(d-commit2). First, we observe that the preconditions to δ_E are satisfied by $\alpha_M(t, q, N_n)$ because the preconditions to δ_E are equal to the preconditions to δ_D the the preconditions to δ_8 are equal to the preconditions to δ_7We have already shown that if (s, p, N_n) satisfy δ_7 then (s, p, N_n) satisfy δ_D and we also conclude that (s, p, N_n) satisfy δ_E if (s, p, N_n) satisfy δ_8.

The postcondition to transition δ_8 has two effects on N'_n: $q := q[s_f \leftarrow s_t]$ as in transition $\delta_7$and, and $\text{next}(q, s) := \text{append}(\text{next}(q, s), s_f)$. Before δ_8 the path abstraction of q, q' is:

$$\psi_M(q, q', N_n) = \tau t_0 \ldots \tau t_n, \tau s_f, \tau t'_0 \ldots \tau t'_n$$

The path abstraction of q,q' after δ_8 is

$$\psi_M(q, q', N'_n) = \tau t_0 \ldots \tau t_n, \tau s_t, s_f, \tau t'_0 \ldots \tau t'_n$$

There are two subcases to consider, (1.a) s_t appears in $\tau t_0 \ldots \tau t_n$ and (1.b) s_t does not appear in $\tau t_0 \ldots \tau t_n$. Case (1.a): if s_t appears in $\tau t_0 \ldots \tau t_n$ then replacing s_f in q with s_t creates a new committed copy of s in q. Since ψ_M keeps only the newest committed copy of a transaction in path, the copy of s_t in $\tau t_0 \ldots \tau t_n$ does not appear in p in $\alpha_M N'_n$. This effect of δ_8 on $\alpha_M N'_n$ is mirrored by the first replacement, $s_t \leftarrow s_f$, in the postcondition of δ_E. Appending a new copy of s_f to q' affects $\psi_M(q,q',N_n)$ only by adding a new copy of s_f to p immediatly after s_tThe replacement of s_f by s_t in q and the addition of s_f to q' are mirrored by the second replacement, $s_f \leftarrow (s_f, s_t)$, in transition δ_E.

Case (1.b): If s_t does not appear in $\tau t_0 \ldots \tau t_n$, then the effects of δ_8 on $\alpha_M N'_n$ are also modeled by δ_E because the replacement of s_t by ε has no effect on the next abstract state.

Case (2): If $p \neq p'$ then the effects of δ_8 on p are the same as in case (1). But if $p \neq p'$ then the new copy of s_f appened to q' appears in p' rather than p as before. These effects are mirrored by abstract transition δ_F. Transition δ_F requires only that s_f is at the head of p. State N_n satisfies this precondition because s_f is at the head of q and q is the last queue in path p (otherwise $p = p'$) so that $\psi_M N_n$ places s_f at the head of path p. The first effect of δ_8 on $\alpha_M N'_n$ is to replace s_f in path p with s_t. Since only the newest copy of s_t is preserved by ψ_M, all previous copies of s_t in p do not appear in $\alpha_M N'_n$. This is modeled in δ_F by replacing s_t with ε in p and replacing s_f with s_t. The second effect of δ_8 on N'_n

is to append a new copy of s_f to the end of q'. In $\alpha_M N'_n$, this places a new copy of s_f on the end of path p'. This effect is mirrored in δ_F by appending s_f to p'.

Rule 9 d-latch. Let $N'_n = \delta_9(q, t, N_n)$ for some queue q and some transaction t and suppose that (q, t, N_n) satisfy the preconditions to δ_9. If t is not a significant transaction, then $\alpha_M N'_n = \alpha_M N_n$ because appending a new copy of t_f to next(q, t) does not change $\alpha_M N'_n$ since neither t nor t_f appear in $\alpha_M N'_n$.

Suppose t is a significant transaction labeled s and that q lies on path p in $\alpha_M N_n$. The effect of δ_9 on $\alpha_M N'_n$ is to append a copy of s_f to the next queue, q', on the path from q to the destination of s. Let q' lie on path p' and we consider the cases (1) $p = p'$ and (2) $p \neq p'$.

In case (1), the effect of δ_9 on $\alpha_M N'_n$ is duplicated by PCI$_A$ rule d-latch: $\delta_G(p, t, \alpha_M N_n)$. First, we show that $\alpha_M N_n$ satisfies the preconditions to δ_G if (q, t, N_n) satisfies the preconditions to δ_9. The precondition to δ_G includes four clauses:

- $s_t \in p$: Satisfied becuase $t_t \in q$ and $\psi_M(q, N_n)$ places all significant transactions from q on path p—including s_t.
- $s \notin$ next(p, s): Because (q,t,N_n) satisfy the preconditions to δ_9, neither t_t nor t_f appear in next(q, t). Since $t \notin$ next(q, t), t is also not a member of any other queue from q to the destination of t [JHM99] and $\psi_M(q'', N_n)$ for all queues q'' betwen q and the destination of t does not place any copies of s in next(p, s).
- compl$(t) \notin$ opp(p): Because (q,t,N_n) satisfy the preconditions to δ_9, compl$(t) \notin$ opp(q). Since compl$(t) \notin$ opp(q), compl(t) does not appear anywhere in the network [JHM99] and $\psi_M(q'', N_n)$ for all q'' in opp(p) does not place a copy of compl(t) in opp(p).
- compl$(t) \notin$ opp(next(p)): Same arguement as compl$(t) \notin$ opp(p).

Next, we show that $\alpha_M N'_n = \delta_G(p, s, \alpha_M N_n)$. Rule δ_9 places a new copy of s_f in next(q, s), so that $\psi_M(q, q', N'_n)$ is

$$\psi_M(q, q', N'_n) = \tau t_0 \ldots \tau t_n, \tau s_t, \tau s_f, \tau t'_0 \ldots \tau t'_n$$

which changes p by placing a new copy of s_f immediatly following s_t because q and q' both on path p. This effect is duplicatied in δ_G by replacing s_t with s_t,s_f in p. Rule δ_9 makes no other changes to N_n so for case (1) we have that $\alpha_M N'_n = \delta_G(p, s, \alpha_M N_n)$

Next, we consider case (2). If $p \neq p'$ then δ_9 places a new copy of s_f in p', the next path after path p, in $\alpha_M N'_n$. This effect on $\alpha_M N'_n$is duplicated by PCI$_A$ rule d-latch-path: $\delta_H(p, s, \alpha_M N_n)$. First, we show that if (q, t, N_n) satisfies the preconditions to δ_9, then $(p, s, \alpha_M N_n)$ satisfies the preconditions to δ_H. The first condition, $p = t_1 \ldots t_n, s_f$, is satisfied because s_f is at the head of the last queue in path p. Transaction s_f is at the head of q because (q, s, N_n) satisfies the preconditions to δ_9, and q is the last queue in p because next(q, s) is on path p' and $p \neq p'$. Since s_f is at the head of the last queue in p, $\psi_M(q, N_n)$ places s_f at the head of p. The other three preconditions to δ_H are identical to the last three preconditions to δ_H and are similarly satisfied. Next, we show that $\alpha_M N'_n = \delta_G(p, s, \alpha_M N_n)$. Rule δ_9 places a new copy of s_f in next(q, s), so that $\psi_M(q, q', N'_n)$ is

$$\psi_M(q, q', N'_n) = \tau t_0 \ldots \tau t_n, \tau s_t, \tau s_f, \tau t'_0 \ldots \tau t'_n$$

as before. However, since $p \neq p'$ the new copy of s_J in q' now appears at the end of path p'. This effect is duplicated in δ_H by appending s_J to p'. Rule δ_9 makes no other changes to N_n so for case (2) we have that $\alpha_M N'_n = \delta_H(p, s, \alpha_M N_n)$ and for any $N'_n = \delta_9(q, t, N_n)$ there exists a PCI_A rule δ_x such that $\alpha_M N'_n = \delta_x(p, s, \alpha_M N_n)$ for some path p and significant transaction s.

Rule 10 d-end. Suppose that $N'_n = \delta_{10}(q, t, N_n)$ for transaction t in queue q and that (q, t, N_n) satisfied the preconditions to δ_{10}. Transition δ_{10} appends the completion of t to the end of the opposite queue. If $D(a, b, t)$ is not a significant transaction, then compl(t) is also not a significant transaction and neither t nor compl(t) appear in the abstraction of the next state so that $\alpha_M N'_n = \alpha_M N_n$.

Next, suppose t is a significant transaction with completion compl(t), labeled s and compl(s), and that q lies on significant path p. In this case, compl(t) appears in opp(p) in $\alpha_M(N'_n)$. The PCI_A d-end rules (I, J or K) mirror the effect of δ_{10}on $\alpha_M(N'_n)$. First, we show that $\alpha_M(q, t, N_n)$ satisfies the preconditions to δ_I,$\delta_J$or δ_K assuming that (q, t, N_n) satisfies the precondition to δ_{10}. The preconditions δ_I,$\delta_J$and δ_K share the same three clauses:

- $p = t_0, \ldots t_n, s_t$. In N_n, s_t at the head of q, which is the last queue on on path from the source of s_t to the target of s_t (otherwise, address(s) $\neq q$). Therefore $\psi_M(s_t, q, N_n)$ places s_t at the head of path p.
- compl(s) $\notin$ opp(p): Because s appears in queue q with s addressed to q, and the completion does not appear in the opposite q, then completion of s does not appear anywhere in the network [JHM99]. The abstraction function never adds transactions, so compl(s) does not appear in M_n either – including path opp(p).
- $addr(s) = p$: s is addressed to queue q in N_n and q lies on path p, so s is addressed to path p in the abstraction of N_n.

Having shown that the preconditions to δ_I,δ_J and δ_K are satisfied by $\alpha_M N_n$, we now show that for every $\alpha_M(N'_n)$ potentialy created by δ_{10}, at least one of δ_I,δ_I or δ_K applied to $\alpha_M N_n$ results in a new abstract state equal to $\alpha_M N'_n$. Transition δ_{10} has three effects on N'_n: first, s_t in q is deleted, second, s_t is added to C and third, compl(t) is appended to opp(q). Let $q' = \text{prev}(q, s)$ with q' lying on path p', and consider two cases: (1) $p = p'$and (2) $p \neq p'$. In each case, appending s_t to C is mirrored by appending s_t to C_α. The other two effects on $\alpha_M N'_n$ are different in each case.

Case (1): $p = p'$. Within case (1), there are two additional subcases: (1.a) $q = s_t$ and (1.b) $q = t_0, \ldots, t_n, s_t$. In subcase (1.a), s_t is the only transaction in in q and s_tSuppose that s_t' is at the head of q'. The path abstraction of q,q' in state N_n is:

$$\psi_M(q, q', N_n) = \tau t'_0 \ldots \tau t'_n, \tau s'_t, \tau s_t$$

which includes only one copy of s_t in p because only the newest committed copy of s is kept in $\psi_M N_n$. Transition δ_{10} deletes s_t from q which under ψ_M, reults in path p with the same transactions because the s_t' in q' is now the next newest copy of s_t. However, transition δ_{10} also appends a new copy of compl(t) on opp(q) which changes the path abstraction of opp(p) by appending compl(t) on opp(p). These effects are mirrored by abstract transaction δ_K. The related subscase in which $q = (s_t)$ but $q' = (t'_1 \ldots t'_i, s_t, t'_{i+1}, \ldots t_n)$. is handled in the next subcase.

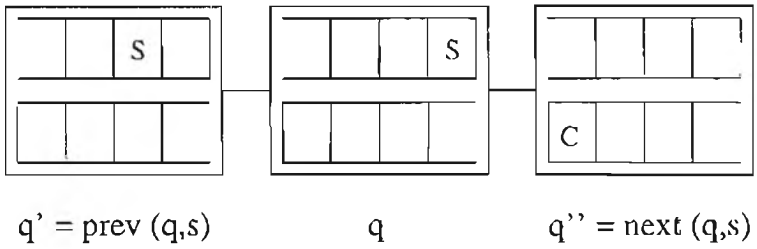

Figure 4: Relative locations of q, q' and q''.

Next, consider subcase (1.b) in which $q = (t_1, \ldots, t_n, s_t)$ and $q' = (t'_1 \ldots t'_n, s_t)$. The path pabstraction of q,q' in state N_n is:

$$\psi_M(q, q', N_n) = \tau t'_0 \ldots \tau t'_n, \tau s'_t, \tau t_0 \ldots \tau t_n, \tau s_t$$

This time, deleting s_t from q changes $\psi_M(q, q', N'_n)$

$$\psi_M(q, q', N'_n) = \tau t'_0 \ldots \tau t'_n, \tau s'_t, \tau t_0 \ldots \tau t_n$$

to include s_t' after the transactions that appear in $\tau t_0 \ldots \tau t_n$. However, since queue boundaries are ignored in $\alpha_M N'_n$, the abstract transition that mirrors this concrete transition does not "know" which transaction to place s_t behind in p in the next state. Consequently, abstract transisition δ_I considers all interleavings of s_t in the transactions in the next state of p. The second effect of δ_{10} on $\alpha_M N'_n$ is to append a compl(s) to the end of opp(p). This effect of δ_{10} is mirrored by δ_I by also appending compl(s) to opp(p). Also, abstract transition δ_J covers the related subcases where $q = (s_t)$ or $q = (t_0, \ldots, t_n, s_t)$ and $q' = (t'_1 \ldots t'_i, s_t, t'_{i+1}, \ldots t_n)$ because queue boundaries are lost in $\alpha_M N'_n$ and once again δ_J does not "know" beforehand where to place s_t in the next state of path p.

Case (2): $p \neq p'$. If $p \neq p'$ then q is the last queue in path p. If q is the last queue in path p, then deleting s_t from q removes the last copy of s_t in p and the only effects of δ_{10} are to remove s_t from p and to append compl(t) to opp(p). Both of these effects are mirrored in the postcondition to abstract rule δ_J.

Rule 11 d-comp. Suppose that (q,t, N_n) satisfy the precondition to δ_{11}

$$q = (t_1 \ldots t_n, D(a, b, t)) \wedge C(a, b, f) \in \text{opp}(\text{next}(q, D(a, b, t)))$$

$D(a, b, t)$ is either a significant or an insignificant transaction. If $D(a, b, t)$ is not a significant transaction, then the abstraction of the next state, $\alpha_M(N'_n)$, is equal to the abstraction of N_n because neither $D(a, b, t)$ nor its completion appear in either abstract state. In this case, the abstract noop rule, δ_S, mirrors the effect of d-comp on $\alpha_M(N'_n)$.

If $D(a, b, t)$ is a significant transactionl labeled s_t, then q lies on a significant path, p, and $\alpha_M(N'_n) \neq \alpha_M(N_n)$. Let $q' = \text{prev}(q, s_t)$ and $q'' = \text{next}(q, s_t)$ with q' on path p' and q'' on path p''. Figure 4 depicts the relative locations of q, q' and q''. In the figure, q contains a copy of s_t as does q' and q'' contains a copy of compl(t). Other transactions that might appear in the queues are ommitted for clarity. There are four cases to consider, and four abstract transitions (δ_L, δ_M, δ_N and δ_O) will be used to mirror the effects of each of the four cases.

First, suppose $p = p' = p''$, in this case we use abstract rule L. To use δ_L we need to show that $\alpha_M(N_n)$ satisfies the preconditions to rule L given that $p = p' = p''$ and N_n satisfies the preconditions to δ_L. There are two preconditions to δ_L:

- $p = (t_0 \ldots t_i, s_t, t_{i+1}, \ldots, t_n)$: Since $t_t \in q$ and $q \in p$, the path abstraction function ψ_M places t_t on path p.
- compl$(t) \in$ opp(p): Since $compl(t) \in$ opp(q) and opp$(q) \in$ opp(p), the path abstraction function ψ_M also places compl(t) on path opp(p).

Transition δ_{11} creates N'_n by updating the state of N_n in three ways which affect the abstract state and are mirrored by abstract transition δ_L. First, $q := q[t_t \leftarrow \varepsilon]$. Suppose that in state N_n, we have:

$$\begin{array}{l} q' = (t'_0, \ldots t'_i, s'_t, t'_{i+1}, \ldots t'_n) \\ q = (t_0, \ldots t_n, s_t) \\ \psi_M(q', \psi_M(q, M')) = \begin{array}{l} (\tau(t'_0), \ldots \tau(t'_i), \tau(s'_t), \tau(t'_{i+1}), \ldots \tau(t'_n)) \\ (\tau(t_0), \ldots \tau(t_n), \tau(s_t)) \end{array} \end{array}$$

which eliminates s'_t because s_t in q is a newer copy of the committed transaction s. However, in state N'_n transaction s_t in q is deleted and s'_t in q' becomes the newest copy of the committed transaction s. The abstraction of these two queues is now:

$$\psi_M(q', \psi_M(q, M')) = \begin{array}{l} (\tau(t'_0), \ldots \tau(t'_i), \tau(s'_t), \tau(t'_{i+1}), \ldots \tau(t'_n)) \\ (\tau(t_0), \ldots \tau(t_n)) \end{array}$$

in which the transactions $\tau(t'_{i+1}), \ldots \tau(t'_n))(\tau(t_0), \ldots \tau(t_n)$ now appear before s'_t in the abstraction of N'_n because s'_t is now the newest committed copy of s. However, when δ_L is applied to the abstraction of N_n, we do not know a priori which transactions should appear before s'_t so in the next abstract state, M'_n, we must consider all interleavings of s'_t in the transactions behind s_t in M_n. One of these interleavings will match the actual location of s_t in $\alpha_M(N'_n)$. The other two affects of δ_{11} on N_n move the completion from the front of opp(q'') to the opp(q). These changes have no affect on $\alpha_M(N'_n)$ because q and q'' are on the same path and moving a transaction from the head of one queue to the tail of the next in the same path has no affect of the abstract state.

Next, suppose $p = p'' \neq p'$ or the previous queue is on a different path and q and the next queue are on the same path. This case is mirrored by abstract transition δ_M (d-comp2). The preconditions to δ_M are satisfied by $\alpha_M N_n$ because the preconditions to δ_M are the same as the preconditions to δ_L. However, in this case the postcondition is different because the next newest significant transaction now appears in path p' in $\alpha_M(N'_n)$ because s_t was in the last queue in path p (otherwise, the previous queue q' would be on the same path as q). Transition δ_M mirrors this effect on $\alpha_M N'_n$ by removing s_t from p.

We now consider the case that $p = p' \neq p''$ or the previous and current queues are on the same path, but the next queue is on the next path. In this case we use rule N (d-comp-next) to mirror the changes in the abstract state space. First, we show that the abstraction of N_n satisfies the preconditions to δ_N if N_n satisfies the precondition to δ_{11} and $p = p' \neq p''$.

- $p = t_0 \ldots t_n, s_t$. Satisfied because s_t is at the head of q and q is the last queue in path p (otherwise, $p = p''$) so $\psi_M(q, M)$ places s_t at the end of path p.

- $\text{opp}(p'') = (\text{compl}(s), t_1 \ldots t_n)$. Satisfied because the completion of s_t appears at the head of q'' and q'' appears at the front of path p'' (otherwise, $p = p''$) so $\psi_M(q, M)$ places s_t at the front of p''.
- $s_t \notin p''$ In a PCI network, if a completion appears in a queue, then the transaction matching that completion does not appear in the opposite queue [JHM99]. Also, if a delayed committed transaction does not appear in a queue, then the transaction does not appear in any queues between that queue and the destination. This precondition is satisfied because the $\text{compl}(s_t)$ appears in $\text{opp}(q'')$ so s_t can not appear in q''. Since $s_t \notin q''$, s_t is also not in any other queues in path p'' and $\psi_M(\hat{q}, M)$ for all $\hat{q} \in p''$ does not place s_t in p''.

When $p = p' \neq p''$, the changes to $\alpha_M(N'_n)$ made by δ_{11} are mirrored by rule N. First, the copy of s_t in q is deleted which moves the next newest copy of s_t from q to q' and changes $\psi_M(q, \psi_M(q', M))$ by allowing the next-newest copy of s_t to appear between any two transactions after s_t in p—as before. Second, the completion of s_t moves from $\text{opp}(q'')$ to $\text{opp}(q)$ which in $\psi_M(q'', \psi_M(q, M))$ moves the completion from path $\text{opp}(p'')$ to path $\text{opp}(p)$. These changes are modeled by δ_N which moves the completion for s_t from $\text{opp}(p'')$ to $\text{opp}(p)$ and allows s_t to appear between any two transactions in p.

Finaly, if $p \neq p' \neq p''$ then abstract rule δ_O (d-comp-next2) mirrors the effects of δ_{11} on $\alpha_M N'_n$. The preconditions to δ_O are the same as the preconditions to δ_N and are satisfied by the same arguement as for δ_N. In this case, δ_{11} deletes the last copy of s_t from p and the next-newest copy of s_t appears in path p' in $\alpha_M N'_n$. Path p does not contain any more copies of s_t because q is the last (and only) queue in p (otherwise $p' = p$) and a queue can contain at most one copy of a delayed transaction [JHM99]. Transition δ_O mirrors this effect by deleting s_t from p. The movement of the completion by δ_{11} and δ_O is the same as for δ_N.

We have shown that in all four cases of d-comp, $\alpha_M(N'_n)$ can be produced from $\alpha_M N_n$ by applying an abstract transition rule.

Rule 12 p-complete. Suppose that (q,t,N_n) satisfy the precondition to transition δ_{12}:

$$(q = (t_1 \ldots t_n, P(a, b, f)))$$

Suppose $P(a, b)$ is not a significant transaction, then $\alpha_M(N_n) = \alpha_M(N'_n)$ because $P(a, b)$ does not appear in either state. Next, suppose $P(a, b) = s$ for some significant transaction s, then the application of δ_{12} may result in a new abstract state if $q' = \text{next}(q, s)$ lies on a different path than q. Let q lie on path p and q' lie on path p'.

First, suppose $p = p'$, then

$$\begin{aligned} &q = t_0 \ldots t_n, s \\ &q' = t'_0 \ldots t'_n \\ &\psi_M(q, \psi_M(q', M)) = p = \tau_M(t_0) \ldots \tau_M(t_n), \tau_M(s), \tau_M(t'_0) \ldots \tau_M(t'_n) \end{aligned}$$

The postcondition to δ_{12} copies s from the head of q to the tail of q', which has no effect on the path abstraction of q, q'

$$\begin{aligned} &q = t_0 \ldots t_n \\ &q' = s, t'_0 \ldots t'_n \\ &\psi_M(q, \psi_M(q', M)) = p = \tau_M(t_0) \ldots \tau_M(t_n), \tau_M(s), \tau_M(t'_0) \ldots \tau_M(t'_n) \end{aligned}$$

Next, suppose $p \neq p'$. In this case, abstract transition δ_Q (p-complete) mirrors the effects of δ_{12} on $\alpha_M(N'_n)$. First, we show that the precondition to transition δ_Q is satisfied by $\alpha_M(q, t, N_n)$ if (q, t, N_n) satisfied the precondition to rule 12 and $p \neq p'$. Transition δ_Q requires that $p = t_0 \ldots t_n, s$ this requirement is satisfied because s is at the head of q, q is the last queue in path p (otherwise, q' would also be on path p and $p = p'$) and ψ_M preserves the order of significant transactions on a path so s is the last transaction on path p.

We now show that the effects of δ_{12} on the abstraction of the next state are mirrored by δ_Q on $\alpha_M(N_n)$. Transition δ_{12} has two effects on N_n:

- $q := q[s \leftarrow \varepsilon]$. This postcondition removes s from q which also removes s from p in the abstract state. A posted transaction may appear in at most one bridge in any network state [JHM99], so removing s from q leaves no copies of s in path p. This affect is mirred by $p := t_0 \ldots t_n$ in δ_Q.
- $q' = \text{append}(s, q')$. This postcondition places s at the end of q', which places s at the end of path p' in the abstract state. This is because q' is the first queue in p' (otherwise, $p = p'$) so placing s at the end of q' places s at the end of path p'. This affect is mirrored by $p' = \mathit{append}(s, p')$ in δ_Q.

All changes in $\alpha_M(N'_n)$ are mirrored by a rule on $\alpha_M(N_n)$, and for all transitions δ_i on N_n, some transition δ_a exists on $\alpha_M(N_n)$ such that

$$\alpha_M(\delta_i(N_n)) = \delta_a(\alpha_M(N_n))$$

from which we conclude that all PCI_A traces created by some concrete PCI trace can be created by an application of the PCI_A transition rules.

Rule 13 p-end. Suppose that (q,t,N_n) satisfy the precondition to transition δ_{13}:

$$q = (t_1 \ldots t_n, P(a, \text{address}(q), f))$$

Transition δ_{13} removes t from q and appends t to the completion list C. If t is not a significant transaction, then $\alpha_M N'_n = \alpha_M N_n$ because t never appeared in $\alpha_M N_n$ so removing t from N_n has no effect on $\alpha_M N'_n$.

If t is a significant transaction, labeled s, then the effects of δ_{13} on $\alpha_M N'_n$ are mirrored by abstract transition δ_R (p-end). First we show that the preconditions to δ_R are satisfied by $(s, p, \alpha_M N_n)$ if (t, q, N_n) satisfy the preconditions to δ_{13}. Transition δ_R has two preconditions:

- $p = t_0 \ldots t_n, s$: Since s is at the head of q and q is the last queue in path p (otherwise, address(q) $\neq q$ because PCI transactions can only be addressed to agents [JHM99]), $\psi_M(s, N_n)$ places s at the end of path p.
- $s = (P, \_, p)$: Transaction s is addressed to p because it is addressed to q and q is the address of p.

Transition δ_{13} has two effects on $\alpha_M N'_n$: δ_{13} deletes s from p and adds s to C_a. Transition δ_{13} deletes s from p because δ_{13} deletes s from q and no other copies of s appear in queues contained in path p. Transition δ_{13} adds s to C_a because δ_{13} adds s to C and the abstraction of C retains s because s is a significant transaction. Both of these effects are mirrored in the postconditions of δ_R.

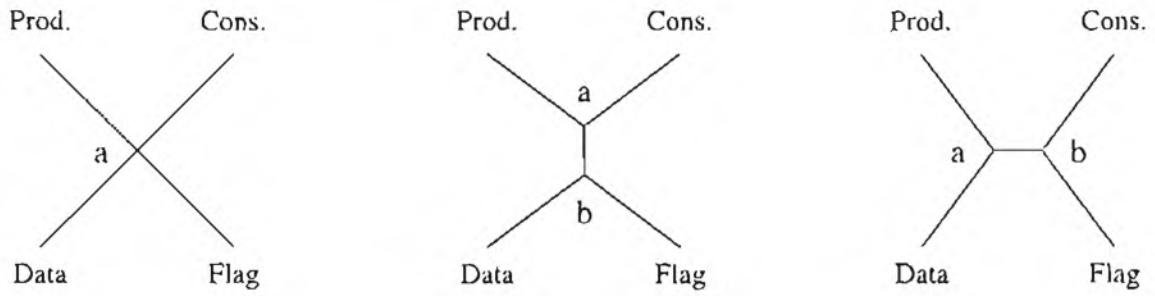

Figure 5: Three classes of producer/consumer PCI networks.

5 PCI_A satisfies producer/consumer

Having shown that PCI is a refinement of PCI_A, we now turn our attention to showing that all traces of PCI_A for all PCI_A network topologies satisfy the producer/consumer property. First, we identify all PCI_A network topologies and second we check all traces in each topology.

A PCI_A network consists of four distinct agents and the paths connecting them. Since PCI_A networks are structural abstractions of PCI networks and PCI networks are defined to satisify certain structural properties, PCI_A networks inherit certain properties that make identifing all PCI_A networks a relatively simple task. PCI networks are defined to have the following properties:

- A unique path exists between every pair of agents. This means that every agents in a PCI_A network is connected by one path to every other agent. This eliminates bifurcations and reconvergent splits because reconvergent splits lead to multiple paths between agents.
- There are no cycles. In PCI_A, the absence of cycles reduces the number of network topologies that must be considered.

Using the properties of PCI networks we have formally proven that all PCI networks with four unique agents fall into one of three classes of networks. These classes of networks are shown in figure 5. The networks in the figure show only the four agents and abstractions of the paths between them. In a real PCI network, each path can contain an arbitrary number of bridges and each network can contain an arbitrary number of agents. This proof was done using the PVS theorem prover and a theory of undirected graphs based on the NASA graph theory library [BS98]. Each class of networks contains an unbounded number of distinct PCI networks because the paths between any two agents may be arbitrarily long and an arbitrary number of other agents may appear in any network. However, all networks in each class have the same abstraction. This means that we can check the canonical abstract network for each class and effectively check all concrete networks in the class. By so doing, we can check all concrete network topologies with four cases.

We checked that all PCI_A traces satisfy the producer/consumer property using an explicit state ennumeration model checker. The PCI_A rules were encoded as they appear in this report. Recall that in PCI_A, a list of completed transactions is kept to track the

Figure 6: Execution times and states reached to check producer/consumer for PCI_A.

Config.	CPU Time	States
A	35.35 sec.	1614
B	18.68 sec.	914
C	12.56 sec.	648
Total	66.59 sec.	3176

order in which transactions were completed. The producer/consumer property was checked using to following predicate on the completion list

```
fun producer_consumer (compl_list) =
if (member (drc,compl_list)) then (member (dw, suffix (drc,compl_list))
    orelse
  (not (member (fw, suffix (frc,compl_list)))))
```

The predicate reads "if the DRC transaction has completed, then either the DW transaction has already completed or the FRC transaction completed before the FW transaction." The three unique producer/consumer topologies were each verified and no violations of producer/consumer were found. The run times and number of states reached for each topology are summarized in table 5.

6 Discussion

Model checking PCI_A has several advantages over model checking PCI. First, PCI_A has fewer states. This means that PCI_A can be checked in less time and space than PCI. We checked PCI_A using an uncompiled MoscowML model checker while we checker PCI using the SPIN model checker. The execution times for the most closely related cases are nearly the same. Second, PCI_A results can be generalized to arbitrary PCI networks with arbitrary transactions. Due to capacity constraints, we limited our previous PCI models to ignore extraneous transactions, agents and bridges. We do not model extraneous transactions, agents and bridges in PCI_A, but we model their potential affects and have a proof that PCI is a refinement of PCI_A.

While the abstraction was generated and validated by hand, the technique sure worked well in this case. Tools based on this approach may appear in the future. Specificaly, tools for doing the abstraction proof and tools for a rule-based safety property model checker that has an expressive input language.

The model of PCI and PCI_A used for the bulk of this report concentrate on the PCI protocol with the completion stealing fix proposed by Corella [Cor96]. It is interesting to consider how our abstraction method would have dealt with completion stealing in the presence of *anonymous completions*—completions without local master IDs added by Corella. The presence of anonymous completions in PCI forces the possible inclusion of completions between any two transactions in PCI_A because extraneous completions not addressed to either of the producer/consumer master agents can not be filtered out by the abstration. Additionaly, in PCI_A any significant committed delayed transaction must

Figure 7: Execution times, states checked and error depths for a model of PCI_A that allows completion stealing

Config.	CPU Time	States	Depth
A	3.59 sec.	68	22
B	3.58 sec.	59	17
C	2.77 sec.	64	18
Total	9.95 sec.	181	

be allowed to complete with any completion that appears in an opposite path. These two modifications to PCI_A add a significant number of states because an anonymous completion may appear at any time between any two transactions. In this model of PCI_A, the initial state has over 200 next states. Such a model is clearly beyond the capacity of our toy MoscowML model checker which can check only 45 states per second in a model with just over a thousand states. Even worse, our implementation slows linearly with the number of next states.

We made several reductions to the inclusion and modeling of anonymous transactions to bring our abstract model within the capacity of our model checker while still allowing completion stealing.

- Anonymous completions can only be added after the flag read has completed at the flag address. This simplification delays the state explosion related to anonymous completions until after the flad read has been completed. If the flag read has not completed before the data read completes, then the preconditions to producer/consumer are violated and the producer/consumer property is trivialy true regardless of the value returned by the data read.
- Anonymous completions are neither reordered nor deleted. This simplification significantly reduces the number of next states for a given state.
- Anonymous completions are added only at the begining and end of a path.

With these reductions, we are able to find a violation of the producer/consumer property in all three network classes in the abstract model. The violations were found in under 30 seconds and produced error traces of less than 25 states. While the trace inclusion refinement presented here does not imply that a violation in PCI_A implies a violation in PCI, the PCI_A error traces suggest corresponding PCI execution traces that violate the producer/consumer property. The time required to find a violation and the depth of each violation are summarized in table 6.

References

[BS98] Ricky W. Butler and Jon A. Sjogren. A PVS Graph Theory Library. Technical Report NASA/TM-1998-206923, NASA Langly Research Center, February 1998.

[Cor96] Francisco Corella. Proposal to fix ordering problem in PCI 2.1, 1996. www.pcisig.com/reflector/thrd8.html#00706.

[CSZ97] Francisco Corella, Robert Shaw, and Cui Zhang. A formal proof of absence of deadlock for any acyclic network of PCI buses. In *Computer Hardware Description Languages, CHDL '97*, Toledo, Spain, April 1997.

[JHM99] Michael Jones, Ravi Hosabettu, and Abdel Mokkedem. Theorems proven about PCI using a PVS theory. Technical Report UU-CS??-99, University of Utah, Department of Computer Science, 1999. DNE. not yet submitted.

[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*, 2000. to appear.

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