PV: An Explicit Enumeration Model-checker *

Ratan Nalumasu and Ganesh Gopalakrishnan

Department of Computer Science
University of Utah, Salt Lake City, UT 84112
{ratan, ganesh}@cs.utah.edu
WWW: http://www.cs.utah.edu/~{ratan, ganesh}

Abstract. PV (Protocol Verifier) is an explicit enumeration based model-checker for verifying finite state systems. It implements a partial order reduction, called Two-phase, and selective caching to combat the state explosion problem faced by model-checkers.

1 Introduction

PV (Protocol Verifier) is an explicit enumeration based model-checker for distributed system protocols. It uses a subset of PROMELA [6], an asynchronous concurrent language ("interleaving semantics") with non-deterministic guarded command, as its modeling language. SPIN [8] is a popular model-checker used for verifying PROMELA models. It implements a partial order reduction algorithm, based on [7, 10], to reduce the size of the state graph of the model without affecting the truth value of the properties. Partial order reductions reduce the size of the state graph by exploiting the fact that in realistic protocols, there are many transitions that "commute" with each other, and hence it is sufficient to explore those transitions in any one order to preserve the truth value of the temporal property under consideration. In essence, from every state, a partial order reduction algorithm selects a subset of transitions to explore, while a normal graph traversal such as depth first search (DFS) algorithm would explore all transitions. Partial order reduction algorithms play a very important role in mitigating state explosion, often reducing the computational and memory cost by an exponential factor.

On a number of examples, we noticed that SPIN does not bring sufficient reductions, and traced the reason to a step of the algorithm called proviso. We formulated a new algorithm, called Two-phase that does not use proviso, and implemented it in a tool called PV. On many practical

* Supported in part by ARPA Order #B990 Under SPAWAR Contract #N0039-95-C-0018 (Avalanche), DARPA under contract #DABT6396C0094 (UV), and NSF MIP MIP-9321836.
examples, we found that PV generates a much smaller graph than SPIN. Another novel aspect of the Two-phase algorithm is that it naturally supports selective caching, which reduces further memory requirements of the model-checker. As the amount of available memory is typically limits the size of the model a model-checker can handle, selective caching may offer an advantage over other partial order reduction algorithms. The details of the algorithm and the proof of correctness of the algorithm can be found in [9]. The tool can be obtained by contacting the authors.

2 Promela Description Language

A Promela description consists of a set of processes that run in an interleaving fashion, and communicate using global variables and communication channels. The interleaving semantics allow one to model the fact that, in most distributed systems, processes run at arbitrary speeds.

A process in Promela consists of a set of local variables and a set of transitions (transitions in Promela resemble statements in C). Promela also provides a special transition ‘assert(expr)’. It is an error for expr to evaluate to 0 when the transition can be executed.

A Promela model also may contain a Büchi automaton specified as a never claim. The never claim represents “bad” paths in the description. Any linear temporal logic (LTL) formulae $\phi$ can be translated into a Büchi automaton $A_\phi$. Hence, the model-checking problem of verifying $M \models \phi$ can be decided by expressing $M$ as a Promela model, and using $A_\neg\phi$ as the never claim.

3 Partial Order Reductions in PV

The major difference between Two phase and other partial order reduction algorithms (including the SPIN algorithm) is the way the algorithms expand a given state. Other partial order reduction algorithms attempt to expand each state visited during the search using a subset of enabled transitions at that state and postponing the rest of the transitions. Algorithm must ensure that no transition is postponed forever, referred to as ignoring problem. To address the ignoring problem, the algorithms use a proviso (or a condition very similar to the provisos). The proviso ensures that when a state $x$ is expanded using only a subset of transitions, then none of resulting states is on the stack maintained by the DFS algorithm that implements the graph exploration algorithm.
Two phase search strategy is completely different: when it encounters a new state \( x \), it first expands the state using only \textit{deterministic} transitions in its first phase resulting in a state \( y \). (Informally, deterministic transitions are the transitions that can be taken at the state without affecting the truth property of the property being verified.) Then in the second phase, \( y \) is expanded \textit{completely}. The advantage of this search strategy is that it is not necessary to use a proviso. As the results in Section 6 show, this often results in a much smaller graph.

4 On-the-fly model-checking in PV

PV is an on-the-fly model-checker, i.e., it verifies if the model contains any violations as it constructs the state graph. PV finds three kinds of errors in the PromeLA model: deadlocks, assertion violations, and \textit{never} claim violations.

Tarjan [11] presented a DFS based on-the-fly algorithm to compute SCCs \textit{without storing any edge information}. Since space is at a premium for most verification problems, not having to store the edge information can be a major benefit of using this algorithm. This algorithm uses one word overhead per state visited and traverses the graph twice.

[4] presents a more efficient on-the-fly model-checking algorithm to find if a graph has at least one infinite path satisfying a Büchi acceptance condition. This algorithm uses 1-bit overhead per state and traverses the graph at most twice. PV uses this algorithm.

5 Selective Caching

When a state \( S_0 \) is expanded in the first phase resulting in states \( S_1, S_2, \ldots, S_n \), a straight-forward implementation would enter all these states into the state graph. However, as presented in [9], it is not necessary to save all states. Instead a state \( S_{i+1} \) can be entered into the state graph if and only if \( S_{i+1} < S_i \) where \( i \in \{1, \ldots, n - 1\} \), where \( < \) is any total order on the reachable states of the model. PV uses bit-wise comparison as \( < \).

6 Experimental Results

Two phase algorithm implemented in PV outperforms the proviso based algorithm implemented in SPIN on examples where there proviso is invoked often, as confirmed by the results in Table 1. This table shows
<table>
<thead>
<tr>
<th>Protocol</th>
<th>Spin</th>
<th>PV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>all</td>
<td>Selective</td>
</tr>
<tr>
<td>B5</td>
<td>243/0.34</td>
<td>11/0.33</td>
</tr>
<tr>
<td>W5</td>
<td>63/0.33</td>
<td>243/0.39</td>
</tr>
<tr>
<td>SC3</td>
<td>17,741/4.6</td>
<td>2,687/1.6</td>
</tr>
<tr>
<td>SC4</td>
<td>749,094/127</td>
<td>102,345/41.0</td>
</tr>
<tr>
<td>Mig</td>
<td>113,628/14</td>
<td>22,805/2.6</td>
</tr>
<tr>
<td>Inv</td>
<td>961,089/37</td>
<td>60,736/5.2</td>
</tr>
<tr>
<td>Pftp</td>
<td>95,241/11.0</td>
<td>187,614/30</td>
</tr>
<tr>
<td>Snoopy</td>
<td>16,279/4.4</td>
<td>14,305/2.7</td>
</tr>
<tr>
<td>WA</td>
<td>4.8e+06/340</td>
<td>706,192/31</td>
</tr>
<tr>
<td>UPO</td>
<td>4.9e+06/210</td>
<td>733,546/32</td>
</tr>
<tr>
<td>ROWO</td>
<td>5.2e+06/330</td>
<td>868,665/44</td>
</tr>
</tbody>
</table>

Table 1. Number of states visited and the time taken in seconds by SPIN and PV on various protocols

results of running SPIN and PV (with and without selective caching enabled) on various protocols. The column "Spin" shows the number of states entered into the state graph and the time taken in seconds by SPIN. The column "all" column in PV shows the number of states entered into the state graph and the time taken in seconds by PV without the selective caching. The column "Selective" in PV shows the number of states entered into the state graph and time taken in seconds by PV with the selective caching. All verification runs are conducted on an Ultra-SPARC-1 with 512MB of DRAM.

Contrived examples: B5 is a trivial system with that shows that the graph generated by PV may be exponentially smaller than the graph generated by SPIN. W5 is a contrived example that shows the converse is also true. W5 has no deterministic states; hence PV degenerates to a full search, while SPIN can find significant reductions. SC is a server/client protocol. This protocol consists of n servers and n clients. A client chooses a server and requests for a service. A service consists of a two round trip messages between server and client and some local computations. SPIN cannot complete the graph construction for n = 4, when the memory is limited to 64MB; when the memory limit is increased to 128MB it generates 750k states.

DSM protocols: Mig and inv are two cache coherency protocols used in the implementation of distributed shared memory (DSM) using a directory based scheme in Avalanche multiprocessor [2]. In a directory based DSM implementation, each cache line has a designated node that acts as its "home", i.e., the node that is responsible for maintaining the co-
herency of the line. When a node needs to access the line, if it does not have the required permissions, it contacts the home node to obtain the permissions. Both mig and inv have two cache lines and four processes; two processors act as home nodes for the cache lines and the other two processors access the cache lines. Both algorithms can complete the analysis of Mig within 64MB of memory, but on inv, SPIN requires 128MB of memory PV on the other hand finishes comfortably generating a modest 27,600 states (with selective caching) or 60,736 states (without selective caching) in 64MB.

**Protocols in SPIN distribution:** Pftp and snoopy protocols are provided as part of SPIN distribution. On pftp, SPIN generates fewer states than PV without state caching. The reason is that there is very little determinism in this protocol. Since PV depends on determinism to bring reductions, it generates a larger state space. However, with state caching, the number of states in the hash table goes down by a factor of 2.7. On snoopy, even though PV generates fewer states, the number of states generated by SPIN and PV (without selective caching) is too close to obtain any meaningful conclusion. The reason for this is two-fold. First, this protocol contains some determinism, which helps PV. However, there are a number of deadlocks in this protocol. Hence, the proviso is not invoked many times. Hence the number of states generated is very close.

**Memory model verification examples:** WA, UPO, and ROWO test the interaction of PA (Precision Architecture from Hewlett-Packard) memory ordering rules with the runway bus protocol [1, 5]. Runway is a high-performance split-transaction bus designed to support cache coherency protocols required to implement a symmetric multiprocessor (SMP). These three protocols consist of two HP PA models connected to the runway bus, executing read and write instructions. These property of interest is whether the PA/runway system correctly implements memory consistency rules called write atomicity (WA), and uniprocessor ordering (UPO), and read-order, write-order (ROWO) [3]. On these protocols, the number of states saved by SPIN is approximately 25 times larger than the number of states saved by PV (with selective caching).

**References**


2. John B. Carter, Chen-Chi Kuo, and Ravindra Kuramkote. A comparison of software and hardware synchronization mechanisms for distributed shared memory