BT: A Bounded Transaction Model Checking for Cache Coherence Protocols*

Abstract

Industrial cache coherence protocols often have too many reachable states, preventing full reachability analysis even for small protocol instances. Several partial search debugging methods are therefore employed, including lossy state compression using hash compaction, and bounded model checking (BMC) which bounds the search depth from initial states. We show that instead of the depth-bounded BMC approach, a bounded transaction (BT) approach is much more effective for debugging cache coherence protocols. The reason lies in the fact that the basic unit of activity in a cache coherence protocol is a transaction — i.e., a set of transitions beginning with a cache agent making a request, and ending with a reply (either positive or negative) supplied by the rest of the system. Therefore, by bounding the number of transactions and the variety of interleaving among transactions in model checking, BT facilitates early detection of bugs for cache coherence protocols. We have built a bounded transaction version for the Murphi [7] model checker, and shown that it can find seeded bugs in protocols far more effectively than BMC, esp. when the latter runs out of memory and misses these bugs.

1 Introduction

Scalable industrial cache coherence protocols are very complex, with their models often spanning 50 or more pages of “rule-style” descriptions written in languages such as Murphi [7], TLA+ [12], or BlueSpec [1]. Even when small instances (e.g., number of cache agents, addresses, etc) of these protocols are model checked, using today’s enumerative model checkers with symmetry reduction[10] or other exact state reduction functionalities, their number of reachable states far exceed that available in today’s large machines.

Abstraction/refinement based verification methods [6, 4], symbolic methods [8, 2], or symbolically assisted methods (e.g., [3]), while showing great promise, have not been demonstrated on industrial coherence protocols. Consequently, a number of limited search methods are in use today for debugging cache coherence protocols. These methods include bounded model checking (BMC) in the enumerative sense of depth-bounded searching, lossy state compression using hash compaction, etc.

We argue that the currently used BMC for debugging cache coherence protocols do not use the available resources wisely. This is because in the case of BMC, every cache agent can make requests whenever possible. On one side, these requests and the corresponding states eat resources in model checking. On the other side, before the replies of these requests are received, such requests and the following transitions usually do not contribute much to detecting bugs. The intuitive reason is that in cache coherence protocols, most coherence properties can only be checked when the copies of a cache line flow in different agents. Therefore, it is possible that BMC cannot explore certain interesting states which lead to bugs before the memory is exhausted, as will be shown in Section 3.

We observe that if the basic unit of work of coherence protocols are tracked, which is a “transaction”, then the memory resources will be far more efficiently utilized. We define the the concept of transaction to be constituted by a staring transition in which a cache agent makes a request, a set of middle transitions (optional) in which the request is being processed by the rest of the system, and an ending transition in which a reply is supplied to the request. Figure 1 shows a simple example of a transaction in a directory-based coherence protocol, where agent-1 initially holds an exclusive cache line: (1) agent-2 sends a request to the directory asking for an exclusive copy; (2) the directory then forwards the request to agent-1, the current owner of the line; (3) in receiving the forwarded request, agent-1 sends the the exclusive copy to the directory and sets itself to be invalid; (4) the directory replies agent-2 with the data from (3)

Figure 1. An example of transaction

The rest of the paper is organized as follows. Section 2 presents the framework of BT, including how to specify transactions for cache coherence protocols and the algorithm used in BT. Section 3 shows the experiment results. Related work and conclusion follow.

2 Bounded Transaction Model Checking

To be able to control transactions in model checking, the first task of BT is to identify the transitions constituting a transaction. After transactions are identified, BT can then decide to start how many new transactions, or guide the current transaction toward finishing itself. This section addresses these problems.

---

*This work was supported in part by an SRC contract (details later).
2.1 Specifying Transactions

As stated before, a transaction in cache coherence protocols is constituted by a starting transition, a set of optional middle transitions and an ending transition. In BT, we only care about starting and ending transitions. Once these transitions are identified in a coherence protocol, all the other transitions in the protocol are deemed as middle transitions. We use this to simply the specification of transactions, because for complex cache coherence protocols, one transition can be in many different transactions, and specifying the exact sequence of middle transitions for each transaction can be non-trivial.

In the Murphi model checker where the idea of BT was implemented, we use weights (integer values) as a simple method to differentiate starting, ending, and middle transitions. Currently, six different weights are supported, ranging from 0 to 5. By default, weight 0 is assigned for every transition, and users can easily feed a different weight in model checking.

As will be shown in the next section, BT always concretely executes every state it explores in model checking. So even if users specify meaningless transactions, BT will never report any false warning. Only the the effectiveness of bug hunting will be affected by different specifications.

![Figure 2. Partial state exploration in BT](image)

2.2 Implementation

The basic idea of BT is for each state $s$, at most “MaxQuota” number of transactions are allowed to be initiated from $s$, and all the possible interleaving among these transactions are explored. Till all the transactions are finished, this constitutes one round of exploration in BT. And the states ending one round become the states starting in the next round. As shown in Figure 2, in each round, every starting state can initiate a transaction which is called a root transaction. Each root transaction can again spawn up to MaxQuota of new leaf transactions in its lifetime, however these leaf transactions can no longer spawn new transactions in the current round. After all the root and leaf transactions are finished, the states in the current round become the starting states in the next round (the starting states in the first round are the initial states generated in model checking). Each starting state in a round can spawn up to MaxQuota of new transactions in its lifetime. Using this method, BT can control the number and the variety of transactions to be explored in one round, thus avoid the blind exploration of every possible interleaving in BMC which can exhaust resources quickly. Figure 3 shows the main algorithm of BT, in which $Q$ represents the state queue for the current round, and $Q'$ for the next round.

```
1: state queue $Q, Q'$;
2: procedure BT
3:   generate initial states and put in $Q'$
4:   round := 0
5:   while round < MaxRound do
6:     $Q := Q' := \emptyset$
7:     while $Q \neq \emptyset$ do
8:       $s := \text{dequeue}(Q)$
9:       generate_next_states($s$)
10:   end while
11:   round++
12: end while
13: end procedure
```

![Figure 3. The main algorithm of BT](image)

To remember the number and the variety of new transactions to be initiated at every state $s$, we added two more attributes in BT for $s$, “quota” and “type”. $s:\text{quota}$ is an integer, meaning how many new transactions are allowed to spawn at $s$. And $s:\text{type}$ is also an integer, meaning the type of the transaction which $s$ is in. In BT, we use the weight of the starting transition of a transaction to represent the type of this transaction.

Based on the quota and type information of each state, BT can selectively choose a subset of enabled rules\(^2\), to generate a set of next states for $s$. Figure 4 shows for each state how BT explores its next states – For each state $s$, if $s:\text{quota} = 0$ then only middle and ending transitions can be fired, thus forcing the current transaction to work toward finishing itself. Otherwise, certain number of new transactions can be spawned. These new transactions are selected based on the type of the current transaction, and how many times each transaction has been explored so far. Figure 5 shows this selection algorithm in which “count” remembers how many times each type of transaction has been explored. This algorithm tries to balance the exploration of each type of transaction in BT, so that as many as possible interesting interleaving can be covered using limited resources in model checking.

Finally, the procedure “fire” in Figure 5 shows how the attributes of the next state of $s$ are set, and when an ending transition is fired on $s$ then next\_s will be put into the state queue for the next round.

3 Experiment Results

We have implemented BT in the Murphi model checker and applied it on two protocols: the Stanford FLASH protocol [11] and the German protocol [9]. The Murphi model

\(^1\)Murphi is a rule-based system, in which a transition is represented as: rule “name" guard $\rightarrow$ action.;

\(^2\)For a rule “guard $\rightarrow$ action”, if $s$ $\Rightarrow$ guard then this rule is said an “enabled” rule at $s$.\n
We injected a bug into this protocol by not marking the com-
state queue
11:
else
if all the invalidation acknowledgments have been collected.

Due to the protocol design, symmetry was not used.

The German protocol used in our experiment models
the FLASH protocol has about 1,100 lines of code. It
has been used as an academic protocol but with industrial
protocol complexity. We injected a bug into the FLASH
protocol to optimize model checking for cache coherence protocols.

The transactional nature of cache coherence protocols
was first described by Park and Dill [13]. They aggre-
gated the implementation step of each transaction into a
single atomic transaction in the specification. Completing
(or “commit”) step was defined as the implementation step
which first causes a change in the specification variables.
 Compared with their approach, BT does not need to know
all the transitions in a transaction, instead only the start-
ing and ending transitions are used. And BT checks the in-
terleaving among transitions in different transactions in the
implementation.

Several other search heuristics have also been proposed
to optimize model checking for cache coherence protocols.
For instance, Yang and Dill[16] proposed using the mini-
um hamming distance as a heuristic with the hope that
states with very few bits differing from the error state will
require fewer cycles to reach the target. However, it is not
clear if these criteria can always lead to effective debugging
for cache coherence protocols.

Recently, Bhattacharya et al.[3] proposed to exploit the
transactional nature of cache coherence protocols to aid par-
tial order reduction. By selecting appropriate seed transi-
tions in a transaction in the ample set computation [5], they can effectively take the current transaction forward and delay the scheduling of new transactions. Compared with their work which uses symbolic methods to do complete state exploration, BT explores partial search concretely and it is not constrained by the limitations of SAT solvers.

The work by Qadeer et al. in [14] is similar with BT. They reduced the model checking of communicating pushdown systems where reachability is inherently undecidable [15], to bounded interleaving checking. Their results show that many bugs are caught in this manner. Although the idea is similar, BT bounds the interleaving by controlling the number of transactions and the variety of interleaving among transaction.

5 Conclusion

We developed BT, a tool to debug cache coherence protocols by bounding the number and the variety of transactions in model checking. The experiment results on the FLASH and German protocol show that BT is a promising debugging tool for coherence protocols.

References


