COMPUTER ARCHITECTURE
Assignment 3: verification-of-cc-protocol

Lakshmi Narasimha Sai Narne (2157472)

Data inconsistency may occur across adjacent levels or within the same level of the memory hierarchy in a system with several processors. In a shared memory multiprocessor with a separate cache memory for each processor, it is possible to have several copies of any single instruction operand: one in main memory and one in each cache memory. When one operand duplicate is edited, the other copies must also be modified.

Cache coherence is a difficulty caused by many processors running in parallel, as well as various caches containing distinct copies of the same memory block. In a multiprocessor system, there are several Cache Coherence Protocols. These are the

1. MSI protocol (Modified, Shared, Invalid)

2. MOSI protocol (Modified, Owned, Shared, Invalid)

3. MESI protocol (Modified, Exclusive, Shared, Invalid)

4. MOESI protocol (Modified, Owned, Exclusive, Shared, Invalid

MODIFIED

It indicates that the value in the cache is dirty, meaning that the value in the current cache differs from the value in main memory.

EXCLUSIVE

It signifies that the value in the cache matches the value in main memory, indicating that the value is clean.

SHARED

It signifies that the cache value contains the most current data copy, which is shared across all cache and main memory.

OWNED

It means that the current cache holds the block and is now the owner of that block that is having all rights on those blocks.

INVALID

This indicates that the current cache block is invalid and must be retrieved from another cache or source. There are three concurrency mechanisms that can be used to solve them.

• Directory Based

• Snooping

• Snarfing

What is Murphi?

Murphi is a well-known tool for testing the correctness of concurrent systems. It enables the user to express the system's behavior in terms of states and transitions, which can subsequently be verified using formal methods. Murphi employs state enumeration to examine all of a system's conceivable states and transitions.

Problems Faced:

There were C and C++ dependencies, and some C libraries could not be used directly in C++. We had to convert the C libraries to C++ for this (like IOSTREAM, NEW etc).

We needed to install certain missing libraries to compile the code in Linux. Some of them are **flex-old**.

Also, I had to install the following package to allow 32 bit libraries to run on my machine.

```
sainarne15@Ubuntu14:~/Downloads/futurebus-master/verification$ sudo apt-get install gcc-multilib g++-multilib
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Execution:

- ➔ Run the make file for mu and install (below are the screenshots of the results)

```
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main$ cd Murphi3.1/src
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/src$ make mu
flex -i mu.l
yacc -vdt mu.y
mu.y: warning: 9 nonterminals useless in grammar [-Wother]
mu.y: warning: 15 rules useless in grammar [-Wother]
mu.y:187.23-29: warning: nonterminal useless in grammar: onerule [-Wother]
```

```
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/src$ make install
rm -f ../bin/mu.
mv mu ../bin/mu.
```

- ➔ Later, we have to create a symbolic link for the runnable mu file. Symbolic links offer a convenient way to organize and share files. They provide quick access to long and confusing directory paths. They are heavily used in linking libraries in Linux. We can create it by using ln –s.
- ➔ After creating it, we can try it on some examples such as the pingpong file. By using the mu, we converted the pingpong to a C file.

```
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/src$ cd ..
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1$ cd bin
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/bin$ ln -sf mu. mu
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/bin$ cd ..
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1$ cd ex
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/ex$ ../bin/mu toy/pingpong.m

========================================================================
Murphi Release 3.1
Finite-state Concurrent System Compiler.

Copyright (C) 1992 - 1999 by the Board of Trustees of
Leland Stanford Junior University.
========================================================================
Call with the -l flag or read the license file for terms
and conditions of use.
Run this program with "-h" for the list of options.
Bugs, questions, and comments should be directed to
"murphi@verify.stanford.edu".

Murphi compiler last modified date: Jan 29 1999
Murphi compiler last compiled date: May  5 2023
========================================================================
Code generated in file toy/pingpong.C
```

➔ In the below, we can see the results for make pingpong

```
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/ex/toy$ make pingpong
g++ -DCATCH_DIV    -o pingpong pingpong.C -I../../include -lm
```

➔ Now we are verifying the pingpong protocol (argument –v should be passed to achieve this) by using various verification algorithms. The default verification method is breadth first search but we can use other verifications such as –s (simulate), -vdfs (using depth first search) etc.

➔ The Output contains Memory Usage, Error Status and the state space explored (number of states and rules).

```
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/ex/toy$ ./pingpong -v
This program should be regarded as a DEBUGGING aid, not as a
certifier of correctness.
Call with the -l flag or read the license file for terms
and conditions of use.
Run this program with "-h" for the list of options.

Bugs, questions, and comments should be directed to
"murphi@verify.stanford.edu".

Murphi compiler last modified date: Jan 29 1999
Include files   last modified date: Jan 29 1999
============================================================================

============================================================================
Murphi Release 3.1
Finite-state Concurrent System Verifier.

Copyright (C) 1992 - 1999 by the Board of Trustees of
Leland Stanford Junior University.

============================================================================

Protocol: toy/pingpong

Algorithm:
        Verification by breadth first search.
        with symmetry algorithm 3 -- Heuristic Small Memory Normalization
        with permutation trial limit 10.

Memory usage:

        * The size of each state is 32 bits (rounded up to 4 bytes).
        * The memory allocated for the hash table and state queue is
          8 Mbytes.
          With two words of overhead per state, the maximum size of
          the state space is 476219 states.
           * Use option "-k" or "-m" to increase this, if necessary.
        * Capacity in queue for breadth-first search: 47621 states.
           * Change the constant gPercentActiveStates in mu_prolog.inc
             to increase this, if necessary.

Warning: No trace will not be printed in the case of protocol errors!
```

```
============================================================================

Protocol: toy/pingpong

Algorithm:
        Verification by breadth first search.
        with symmetry algorithm 3 -- Heuristic Small Memory Normalization
        with permutation trial limit 10.

Memory usage:

        * The size of each state is 32 bits (rounded up to 4 bytes).
        * The memory allocated for the hash table and state queue is
          8 Mbytes.
          With two words of overhead per state, the maximum size of
          the state space is 476219 states.
           * Use option "-k" or "-m" to increase this, if necessary.
        * Capacity in queue for breadth-first search: 47621 states.
           * Change the constant gPercentActiveStates in mu_prolog.inc
             to increase this, if necessary.

Warning: No trace will not be printed in the case of protocol errors!
         Check the options if you want to have error traces.

============================================================================

Status:

        No error found.

State Space Explored:

        4 states, 6 rules fired in 0.10s.
```

Here, we can see that no errors were found after applying the verification by bfs and 4 states, 6 rules fired in 0.10s.

➔ Following is the output for help. We can clearly see all the arguments and their functions.

```
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/ex/toy$ ./pingpong -h
This program should be regarded as a DEBUGGING aid, not as a
certifier of correctness.
Call with the -l flag or read the license file for terms
and conditions of use.

Bugs, questions, and comments should be directed to
"murphi@verify.stanford.edu".

Murphi compiler last modified date: Jan 29 1999
Include files   last modified date: Jan 29 1999
============================================================================
Options:
1) General:
        -h              help.
        -l              print license.
2) Verification Strategy: (default: -v)
        -s              simulate.
        -v or -vbfs     verify with breadth-first search.
        -vdfs           verify with depth-first search.
        -ndl            do not check for deadlock.
3) Others Options: (default: -m8, -p3, -loop1000)
        -m<n>           amount of memory for closed hash table in Mb.
        -k<n>           same, but in Kb.
        -loop<n>        allow loops to be executed at most n times.
        -p              make simulation or verification verbose.
        -p<n>           report progress every 10^n events, n in 1..5.
        -pn             print no progress reports.
        -pr             print out rule information.
4) Error Trace Handling: (default: -tn)
        -tv             write a violating trace (with default -td).
        -td             write only state differences from the previous states.
                        (in simulation mode, write only state differences in
                         verbose mode.)
        -tf             write full states in trace.
                        (in simulation mode, write full states in verbose mode.)
        -ta             write all generated states at least once.
        -tn             write no trace (default).
```

```
5) Reduction Technique: (default: -sym3 with -permlimit 10 and multiset
                                  reduction)
        -nosym          no symmetry reduction (multiset reduction still effective)
        -nomultiset     no multiset reduction
        -sym<n>         reduction by symmetry
        -permlimit<n>   max num of permutation checked in alg 3
                        (for canonicalization, set it to zero)
                        n | methods
                        ---------------------------------
                        1 | exhaustive canonicalize
                        2 | heuristic fast canonicalization
                            (can be slower or faster than alg 3 canonicalization)
                            (use a lot of auxiliary memory for large scalarsets)
                        3 | heuristic small mem canonicaliztion/normalization
                            (depends on -permlimit)
                        4 | heuristic fast normalization (alg 3 with -permlimit 1)
```

For ./pingpong –pr, it printed out the information of rules.

```
============================================================================

Status:

        No error found.

State Space Explored:

        4 states, 6 rules fired in 0.10s.

Rules Information:

        Fired 1 times   - Rule "Pass ball, p:0"
        Fired 1 times   - Rule "Pass ball, p:1"
        Fired 1 times   - Rule "Keep ball, p:0"
        Fired 1 times   - Rule "Keep ball, p:1"
        Fired 1 times   - Rule "Get ball, p:0"
        Fired 1 times   - Rule "Get ball, p:1"
```

For States we need to use ./pingpong -ta

```
State 1:
Players[0].hasball:true
Players[0].gotball:false
Players[1].hasball:false
Players[1].gotball:false

State 2:
Players[0].hasball:false
Players[0].gotball:false
Players[1].hasball:true
Players[1].gotball:false

State 3:
Players[0].hasball:false
Players[0].gotball:false
Players[1].hasball:false
Players[1].gotball:true

State 4:
Players[0].hasball:false
Players[0].gotball:true
Players[1].hasball:false
Players[1].gotball:false
```

Verification using the Future Bus protocol:

For the Verification, We can verify the Murphi by using the futurebus in the verification folder. To use it, we need to go to the directory Murphi3.1/src, and optionally make the executable access using the command chmod +x Murphi3.1/src/mu, if you are in the home directory.

Next we have to go to the verification directory and generate the C file for the Futurebus protocol + verification using ./../Murphi3.1/src/mu futurebus.m command, This will generate the C file futurebus.C.

```
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1/src$ cd ..
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/Murphi3.1$ cd ..
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main$ chmod +x Murphi3.1/src/mu
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main$ cd verification
sainarne15@Ubuntu14:~/Downloads/fixed_murphi3.1-main/verification$ ./../Murphi3.1/src/mu futurebus.m

========================================================================
Murphi Release 3.1
Finite-state Concurrent System Compiler.

Copyright (C) 1992 - 1999 by the Board of Trustees of
Leland Stanford Junior University.
========================================================================
Call with the -l flag or read the license file for terms
and conditions of use.
Run this program with "-h" for the list of options.
Bugs, questions, and comments should be directed to
"murphi@verify.stanford.edu".

Murphi compiler last modified date: Jan 29 1999
Murphi compiler last compiled date: May  5 2023
```

```
futurebus.m:365: warning: Scalarset is used in loop index.
        Please make sure that the iterations are independent.
Code generated in file futurebus.C
```

Copile futurebus.C file using the command make futurebus, which will generate the executable program. Finally we have to run the executable using the following command ./futurebus.

```
sainarne15@Ubuntu14:~/Downloads/futurebus-master/verification$ make futurebus
g++ -std=c++98 -DCATCH_DIV  -Wno-write-strings -m32  -o futurebus futurebus.C -I../Murphi3.1/include -lm
In file included from ../Murphi3.1/include/mu_epilog.inc:79:0,
                 from futurebus.C:3847:
```

The following is the result of the verification of the Futurebus plus cache coherence protocol, which includes a brief overview of any problems or difficulties discovered, as well as other statistics.

```
sainarne15@Ubuntu14:~/Downloads/futurebus-master/verification$ ./futurebus
This program should be regarded as a DEBUGGING aid, not as a
certifier of correctness.
Call with the -l flag or read the license file for terms
and conditions of use.
Run this program with "-h" for the list of options.

Bugs, questions, and comments should be directed to
"murphi@verify.stanford.edu".

Murphi compiler last modified date: Jan 29 1999
Include files   last modified date: Jan 29 1999
============================================================================

============================================================================
Murphi Release 3.1
Finite-state Concurrent System Verifier.

Copyright (C) 1992 - 1999 by the Board of Trustees of
Leland Stanford Junior University.

============================================================================

Protocol: futurebus

Algorithm:
        Verification by breadth first search.
        with symmetry algorithm 3 -- Heuristic Small Memory Normalization
        with permutation trial limit 10.

Memory usage:

        * The size of each state is 96 bits (rounded up to 12 bytes).
        * The memory allocated for the hash table and state queue is
          8 Mbytes.
          With two words of overhead per state, the maximum size of
          the state space is 487811 states.
           * Use option "-k" or "-m" to increase this, if necessary.
```

```
Memory usage:

        * The size of each state is 96 bits (rounded up to 12 bytes).
        * The memory allocated for the hash table and state queue is
          8 Mbytes.
          With two words of overhead per state, the maximum size of
          the state space is 487811 states.
           * Use option "-k" or "-m" to increase this, if necessary.
        * Capacity in queue for breadth-first search: 48781 states.
           * Change the constant gPercentActiveStates in mu_prolog.inc
             to increase this, if necessary.

Warning: No trace will not be printed in the case of protocol errors!
         Check the options if you want to have error traces.

============================================================================

Result:

        Invariant "only one processor in EM state" failed.

State Space Explored:

        56 states, 184 rules fired in 0.10s.

Analysis of State Space:

        There are rules that are never fired.
        If you are running with symmetry, this may be why.  Otherwise,
        please run this program with "-pr" for the rules information.
```

Using the command suffix -tv, we can then publish the output of Futurebus error traces including cache coherence protocol traces. This explains why the mistake occurred (i.e., two

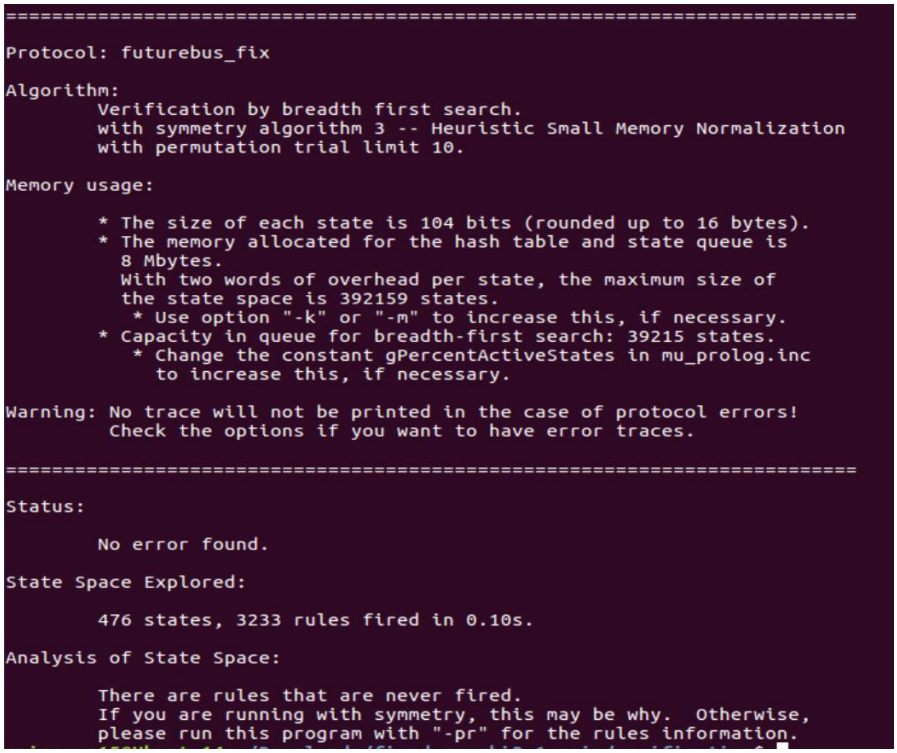
processors enter exclusive states).

```
The following is the error trace for the error:

        Invariant "only one processor in EM state" failed.

Startstate Startstate 0 fired.
proc_state[Proc_1].state:FB_I
proc_state[Proc_1].value:Undefined
proc_state[Proc_2].state:FB_I
proc_state[Proc_2].value:Undefined
proc_state[Proc_3].state:FB_I
proc_state[Proc_3].value:Undefined
transaction_flag:false
last_write:Undefined
one_flag:false
more_flag:false
send_msg.mtype:Undefined
send_msg.value:Undefined
----------

Rule Trying to write data, v:Value_1, i:Proc_1 fired.
proc_state[Proc_1].state:FB_PW
----------

Rule Trying to write data, v:Value_1, i:Proc_2 fired.
proc_state[Proc_2].state:FB_PW
----------

Rule Write data, on DACKemw, v:Value_1, i:Proc_1 fired.
proc_state[Proc_1].state:FB_EM
proc_state[Proc_1].value:Value_1
proc_state[Proc_2].value:Value_1
transaction_flag:true
last_write:Value_1
send_msg.mtype:ReadModified
send_msg.value:Value_1
----------
```

States:

```
State 54:
proc_state[Proc_1].state:FB_EM
proc_state[Proc_1].value:Value_1
proc_state[Proc_2].state:FB_PR
proc_state[Proc_2].value:Undefined
proc_state[Proc_3].state:FB_PW
proc_state[Proc_3].value:Value_1
transaction_flag:true
last_write:Value_1
one_flag:false
more_flag:false
send_msg.mtype:ReadModified
send_msg.value:Value_1

State 55:
proc_state[Proc_1].state:FB_EM
proc_state[Proc_1].value:Value_1
proc_state[Proc_2].state:FB_PW
proc_state[Proc_2].value:Undefined
proc_state[Proc_3].state:FB_PW
proc_state[Proc_3].value:Value_1
transaction_flag:true
last_write:Value_1
one_flag:false
more_flag:false
send_msg.mtype:ReadModified
send_msg.value:Value_1
```

Finally, using the command suffix -pr, we can obtain the output of rule firing of Futurebus + traces of cache coherence protocol to see what rules are fired.



Code: https://github.com/COSC-6385/cache-coherence-model-checking-sainarne15

Instructions can be found in README.md.

References:

1. GitHub - adnaneGdihi/fixed_murphi3.1: fixed_murphi3.1
2. https://linuxhandbook.com/symbolic-link-linux/#:~:text=chained%20symbolic%20link-,What%20is%20Symbolic%20link%20in%20Linux%20and%20why%20is%20it,alias%20to%20an%20actual%20file.
3. https://github.com/Amutheezan/futurebus