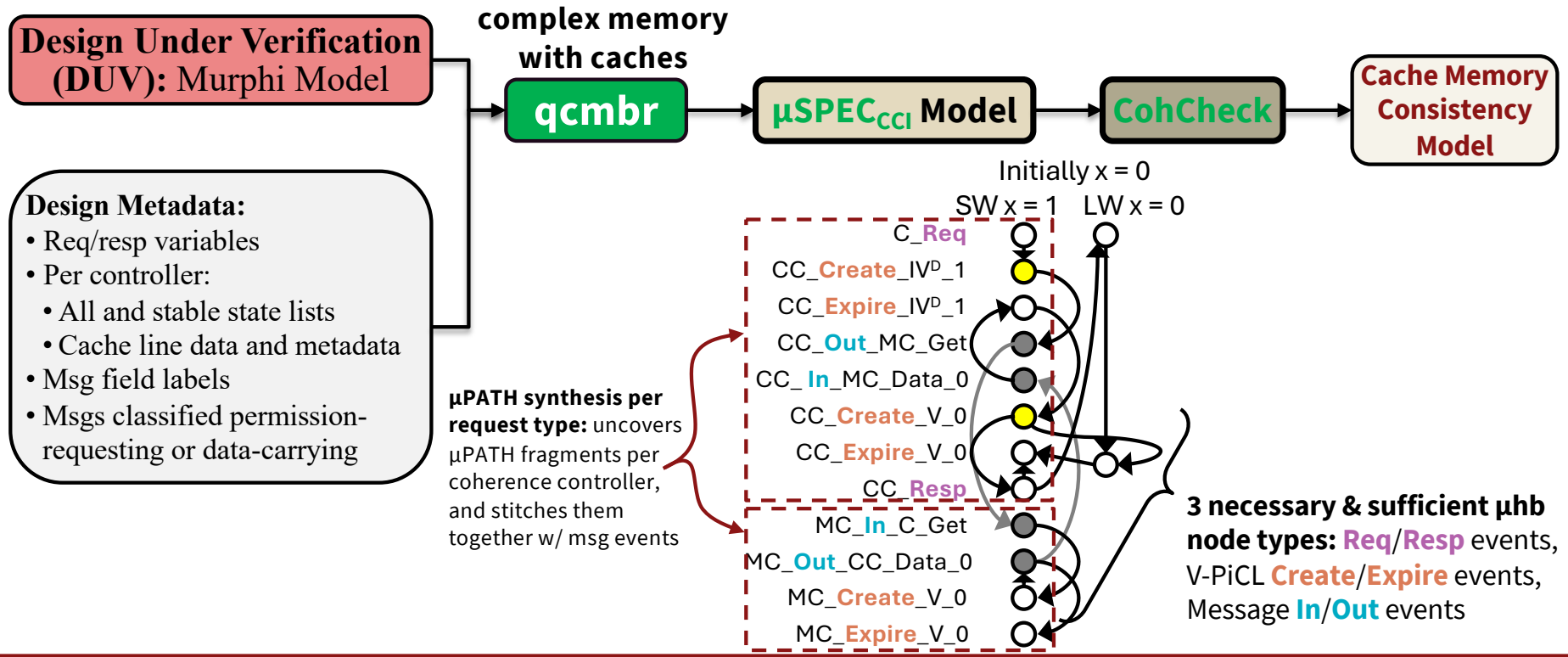
Coherence Verification: QCMBR (Overview)

Caroline

Stanford | ENGINEERING

qcmbr • [Hsiao+, Under Sub.]

Overview: Synthesizing formally verified memory consistency models from Murphi coherence protocols - Complex Memory w/ Caches



Coherence Verification: QCMBR (Hands-On)

Yao

Stanford | ENGINEERING

Roadmap: Apply QCMBR+CohCheck on a buggy VI protocol

- **Step 1: [Input] Annotations**
- Step 2: [Output] Run QCMBR+CohCheck on a buggy VI protocol
- Step 3: [Output] Inspect the verification results and fix the bug
- Step 4: [Output] Re-run CohCheck on the fixed VI $\mu SPEC_{CCI}$ model

In this section, we will work under the qcmbr folder:

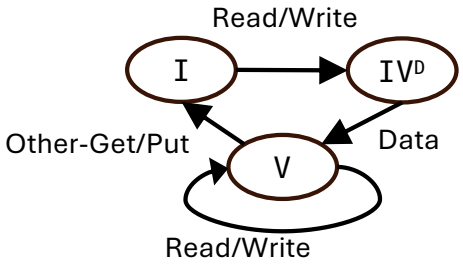
```
$ cd ~/fava_isca2026/qcmbr;
```

[Hands-On] Annotations

Step 1: In the input Murφ model, annotate each point a message is sent or received and supply metadata, including state lists and variable names holding requests, request types, store data values, and load return values

Example: Buggy VI protocol adapted from UMich EECS570 course material

```
$ cd murphi && vim protocols/plain/mi_no_fetch_on_write_buggy.m
```



Cache controller (CC) coherence states

```
46 typ
...
82    ProcStateEnum: enum { P_I, P_IVD, P_V};
83
84    ProcState:
85      Record
86        state: ProcStateEnum;
87        val: Value;
88      End;
...
94 var
...
96    HomeNode:  HomeState;
97    Procs: array [Proc] of ProcState;
...
```

CC struct: state and value

Array of CC structs

`rule` defines transition relation

Invoked when a CC receives load request in I state

```
276    rule "load_P_I"
277      -- only if previous transaction is done
278      (p.state = P_I) & !msgBufNonEmpty &
 IsUndefined(gBuf.src)
279      ==>
280        get_load_req(n);
281        send_req (GetMsg, n, UNDEFINED, UNDEFINED);
282        p.state := P_IVD;
```

Triggering of rule invokes several functions and transitions to IVD

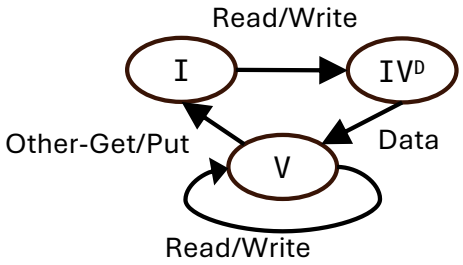
protocols/plain/mi_no_fetch_on_write_buggy.m

[Hands-On] Annotations

Step 1: In the input Murφ model, annotate each point a message is sent or received and supply metadata, including state lists and variable names holding requests, request types, store data values, and load return values

Example: Buggy VI protocol adapted from UMich EECS570 course material

```
$ cd murphi && vim protocols/plain/mi_no_fetch_on_write_buggy.m
```



`rule` defines transition relation

Invoked when a CC receives load request in I state

```
276   rule "load_P_I"
277     -- only if previous transaction is done
278     (p.state = P_I) & !msgBufNonEmpty &
 IsUndefined(gBuf.src)
279     ==>
280       get_load_req(n);
281       send_req (GetMsg, n, UNDEFINED, UNDEFINED);
282       p.state := P_IVD;
```

Triggering of rule invokes several functions and transitions to IVD

```
153 Procedure send_req(mtype:MessageType;
154 src:Node;
155 dst:Node;
156 val:Value;
157 );
...
162   gBuf.mtype := mtype;
163   gBuf.src   := src;
164   gBuf.dst   := dst;
165   gBuf.val   := val;
166   for n:Node do
167     ackBus[n] := false;
168   endfor;
169   -- #SENDING,gBuf
...
```

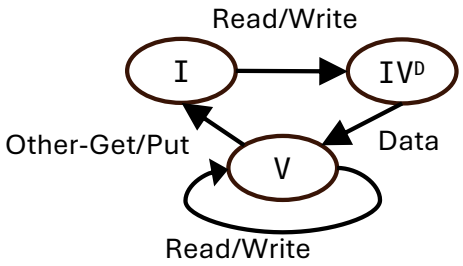
protocols/plain/mi_no_fetch_on_write_buggy.m

[Hands-On] Annotations

Step 1: In the input Murφ model, annotate each point a message is sent or received and supply metadata, including state lists and variable names holding requests, request types, store data values, and load return values

Example: Buggy VI protocol adapted from UMich EECS570 course material

```
$ cd murphi && vim protocols/plain/mi_no_fetch_on_write_buggy.m
```



`rule` defines transition relation

Invoked when a CC receives load request in I state

```
276   rule "load_P_I"
277     -- only if previous transaction is done
278     (p.state = P_I) & !msgBufNonEmpty &
 IsUndefined(gBuf.src)
279     ==>
280       get_load_req(n);
281       send_req (GetMsg, n, UNDEFINED, UNDEFINED);
282       p.state := P_IVD;
```

Triggering of rule invokes several functions and transitions to IVD

Generic message send request (can be invoked by any controller)

```
153 Procedure send_req(mtype:MessageType;
154 src:Node;
155 dst:Node;
156 val:Value;
157 );
...
162   gBuf.mtype := mtype;
163   gBuf.src   := src;
164   gBuf.dst   := dst;
165   gBuf.val   := val;
166   for n:Node do
167
168
169
...
```

Message contains: name, src CC, dst CC, value

protocols/plain/mi_no_fetch_on_write_buggy.m

[Hands-On] Annotations

Step 1: In the input Murφ model, annotate each point a message is sent or received and supply metadata, including state lists and variable names holding requests, request types, store data values, and load return values

Example: Buggy VI protocol adapted from UMich EECS570 course material

```
$ vim src/VI_buggy_gconsts.py
```

```
153 Procedure send_req(mtype:MessageType;
154 src:Node;
155 dst:Node;
156 val:Value;
157 );
...
162   gBuf.mtype := mtype;
163   gBuf.src   := src;
164   gBuf.dst   := dst;
165   gBuf.val   := val;
166   for n:Node do
167     ackBus[n] := false;
168   endfor;
169   -- #SENDING,gBuf
...
```

Annotate the point where message preparation completes

protocols/plain/mi_no_fetch_on_write_buggy.m

Input

Request types

```
 2 all_req_types = ["ci_load", "ci_store", "ci_evict"]
 3 req_is_read = ["ci_load"]
 4 type_with_args = ["ci_store"]
...
16 m_proc_state_type = "ProcStateEnum"
17 m_proc_state_field = "state"
...
34 all_cc_states = ["P_I", "P_IVD", "P_V"]
35 all_cc_stable_states = ["P_I", "P_V"]
```

Variable name for CC's coherence state

All coherence states and stable states

src/VI_buggy_gconsts.

Roadmap: Run QCMBR+CohCheck on a buggy VI protocol

- Step 1: [Input] Annotations
- **Step 2: [Output] Run QCMBR+CohCheck on a buggy VI protocol**
- Step 3: [Output] Inspect the verification results and fix the bug
- Step 4: [Output] Re-run CohCheck on the fixed VI $\mu SPEC_{CCI}$ model

[Hands-On] Roadmap: Run QCMBR+CohCheck on a buggy VI protocol

Step 2: QCMBR's inputs are a Murφ model and metadata; its output is a verified $\mu SPEC_{CCI}$ model. CohCheck conducts litmus-test based evaluation against the synthesized $\mu SPEC_{CCI}$ model.

Example: Coherence verification on synthesized $\mu SPEC_{CCI}$ of the buggy VI protocol

```
$ vim artifact/vi_buggy_test.uarch
$ cd ../check/sandbox/murphi_tests && ./eval.sh run
```

QCMBR Output & CohCheck Input

```
  1 StageName 0 "CReq".
  2 VTStageName 1 0 "cc_P_I_1".
  3 StageName 2 "cc_P_I_lst".
  4 VTStageName 3 1 "cc_P_IVD_1".
  5 StageName 4 "cc_P_IVD_lst".
...
335 Axiom "ci_load_upath":
336 forall microop "i", OnCore c i => (~AccessType
 InitAcc i /\ ~AccessType Evict i /\ IsAnyRead i)=>
337  ((ExpandMacro h_empty_case /\
338 ExpandMacro _c_case_ci_load_0 /\
339 ExpandMacro c_case_label0
340 ) \/
341  (ExpandMacro h_empty_case /\
342 ExpandMacro c_case_ci_load_1 /\
343 ExpandMacro c_case_label_1
344 ) \/
345  (((ExpandMacro h_case_H_I_GetMsg_0  )) /\
346 ExpandMacro c_case_ci_load_2 /\
347 ExpandMacro c_case_label_2
```

artifact/vi_buggy_test.uarch

CohCheck Output

Test	Time	TimeSMT	Bugs	Strict	Inst#
2_2W	0.41	1.22	Yes	No	8
co-iriw	0.58	2.56	No	No	10
co-mp	0.20	1.08	No	No	8
coRR	0.10	0.51	No	No	7
coRW1	0.02	0.03	No	No	3
coRW2_P	0.13	0.55	No	No	7
coRW2	0.13	0.56	No	No	7
coWR_1thd	0.03	0.07	Yes	No	3
coWR2_2	0.07	0.33	No	No	6
coWR2	0.09	0.32	No	Yes	6
coWR	0.14	0.64	No	No	7
coWW_R	0.11	0.61	Yes	No	7
coWW	0.04	0.08	No	No	4
n4	0.51	1.68	No	No	8
n5	0.19	0.58	No	No	6
S_poss	0.30	1.31	No	No	8
...					

result after running eval.sh

[Hands-On] Roadmap: Run QCMBR+CohCheck on a buggy VI protocol

Step 2: QCMBR's inputs are a Murφ model and metadata; its output is a verified $\mu SPEC_{CCI}$ model. CohCheck conducts litmus-test based evaluation against the synthesized $\mu SPEC_{CCI}$ model.

Example: Coherence verification on synthesized $\mu SPEC_{CCI}$ of the buggy VI protocol

```
$ vim artifact/vi_buggy_test.uarch
$ cd ../check/sandbox/murphi_tests && ./eval.sh run
```

QCMBR Output & CohCheck Input

```
  1 StageName 0 "CReq".
  2 VTStageName 1 0 "cc_P_I_1".
  3 StageName 2 "cc_P_I_lst".
  4 VTStageName 3 1 "cc_P_IVD_1"
  5 StageName 4 "cc_P_IVD_lst".
...
335 Axiom "ci_load_upath":
336 forall microop "i", OnCore c i => (~AccessType
 InitAcc i /\ ~AccessType Evict i /\ IsAnyRead i)=>
337  ((ExpandMacro h_empty_case /\
338 ExpandMacro _c_case_ci_load_0 /\
339 ExpandMacro c_case_label0
340 ) \/
341  (ExpandMacro h_empty_case /\
342 ExpandMacro c_case_ci_load_1 /\
343 ExpandMacro c_case_label_1
344 ) \/
345  (((ExpandMacro h_case_H_I_GetMsg_0  )) /\
346 ExpandMacro c_case_ci_load_2 /\
347 ExpandMacro c_case_label_2
```

Beginning (1) of the time of CC in coherence state P_I

End (lst) of the time of CC in coherence state P_I

Symbolic instruction i on symbolic core c

If i is a read request

Disjunction (\/) over a ways in which read req executed by the p

artifact/vi_buggy_test.uarch

CohCheck Output

Test	Time	TimeSMT	Bugs	Strict	Inst#
2_2W	0.41	1.22	Yes	No	8
co-iriw	0.58	2.56	No	No	10
co-mp	0.20	1.08	No	No	8
coRR	0.10	0.51	No	No	7
coRW1	0.02	0.03	No	No	3
coRW2_P	0.13	0.55	No	No	7
coRW2	0.13	0.56	No	No	7
coWR_1thd	0.03	0.07	Yes	No	3
coWR2_2	0.07	0.33	No	No	6
coWR2	0.09	0.32	No	Yes	6
coWR	0.14	0.64	No	No	7
coWW_R	0.11	0.61	Yes	No	7
coWW	0.04	0.08	No	No	4
n4	0.51	1.68	No	No	8
n5	0.19	0.58	No	No	6
S_poss	0.30	1.31	No	No	8
...					

result after running eval.sh

Roadmap: Apply QCMBR+CohCheck on a buggy VI protocol

- Step 1: [Input] Annotations
- Step 2: [Output] Run QCMBR+CohCheck on a buggy VI protocol
- **Step 3: [Output] Inspect the verification results and fix the bug**
- Step 4: [Output] Re-run CohCheck on the fixed VI $\mu SPEC_{CCI}$ model

[Hands-On] Inspect the evaluation result and fix the bug

Step 3: CohCheck leverages μHB analysis, which models hardware-specific program execution as directed graph. **An acyclic graph for a forbidden program execution indicates a bug**.

Example: View the acyclic μHB graph for coWR_1thd

```
$ ./eval.sh view
```

Initial: [x] = 0
C0
i0: ST [x] ← 1
i1: LD [x] → r1
Coherence forbids: r1 = 0

coWR_1thd.test

i0 at row label cc_P_IVD_1

- V-PiCL: Value-Permission pair orchestrated by protocol during request execution
- Value associated with each V-PiCL in terminal output

```
===> v 2 h_H_I_1  has value  [(0, ...
===> v 0 h_H_V_1  has value  [(0, ...
===> v 0 cc_P_IVD_1  has value  [(1, ...
'order_node_g0_c0_0_46', 'order_node_g0_c0_0_29')]
===> v 0 cc_P_V_1  has value  [(0, ....
....
```

Value of 1

i1 at row label cc_P_V_1

Value of 0!

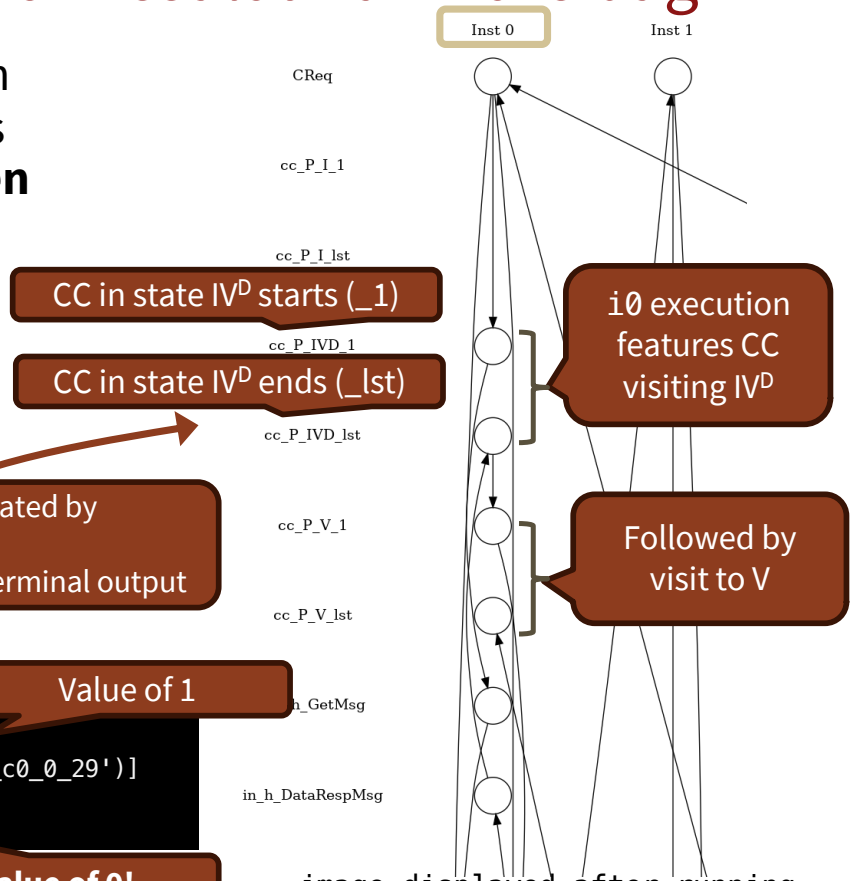


image displayed after running `eval.sh view`

[Hands-On] Inspect the evaluation result and fix the bug

Step 3: CohCheck leverages μHB analysis, which models hardware-specific program execution as directed graph. **An acyclic graph for a forbidden program execution indicates a bug**.

Example: View the acyclic μHB graph for coWR_1thd

```
$ ./eval.sh view
```

Initial: [x] = 0
C0
i0: ST [x] ← 1
i1: LD [x] → r1
Coherence forbids: r1 = 0

coWR_1thd.test

- V-PiCL: Value-Permission pair orchestrated by protocol during request execution
- Value associated with each V-PiCL in terminal output

```
===> v 2 h_H_I_1  has value  [(0, ...
===> v 0 h_H_V_1  has value  [(0, ...
===> v 0 cc_P_IVD_1  has value  [(1, ...
'order_node_g0_c0_0_46', 'order_node_g0_c0_0_29')]
===> v 0 cc_P_V_1  has value  [(0, ....
....
```

result after running `eval.sh view`

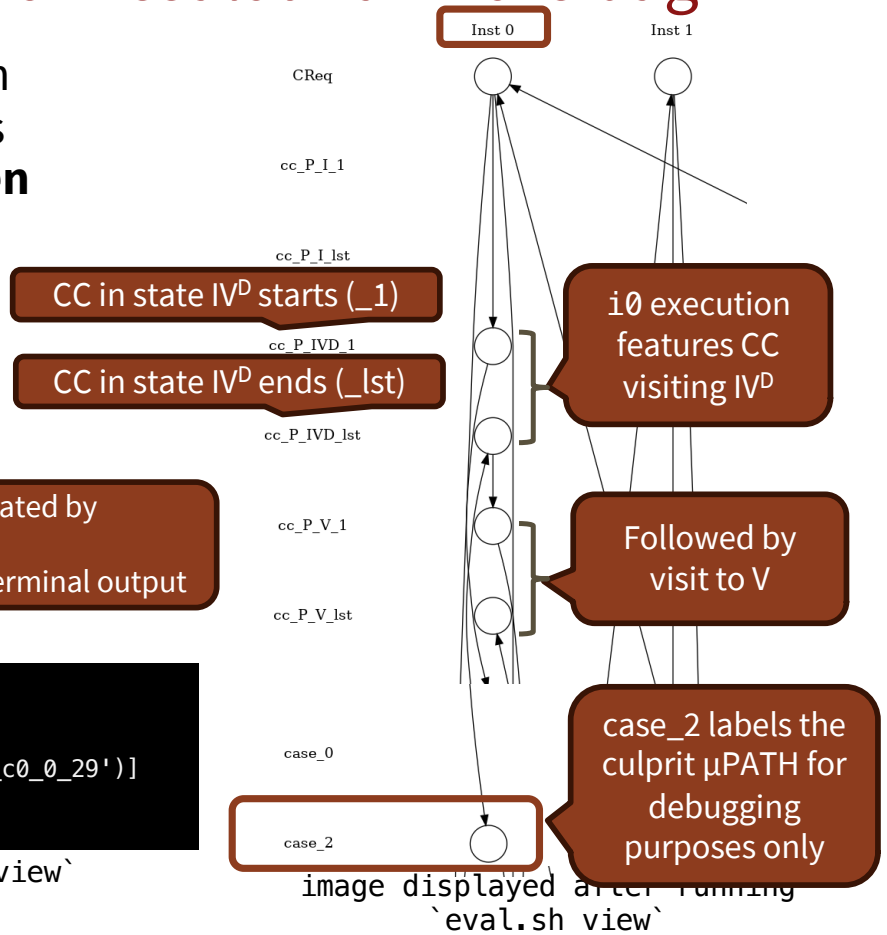


image displayed after running `eval.sh view`

[Hands-On] Inspect the evaluation result and fix the bug

Step 3: QCMBR uses property generation and model checking to produce the synthesized μSPEC$_{CCI}$ model ➔ every expression in the μSPEC$_{CCI}$ model can be traced back to a property evaluation result

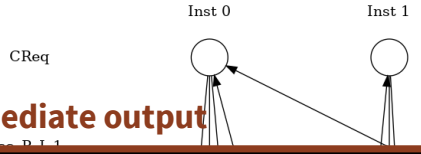
Example: Take a look at the synthesized μSPEC$_{CCI}$ model to debug

```
$ cd ../../../murphi/ && vim -o artifact/vi_buggy_test.uarch
build/s3_2_reachable_set_val/_build/P_I_ci_store_0_val_eq_req_for_P_V.txt
```

QCMBR Output & CohCheck Input

```
116 DefineMacro "c_case_ci_store_2":
117 AddEdge ((i, CReq), (i, MResp), "", "black") /\
118 AddEdges[((i, CReq), (i, cc_P_IVD_1), "to_picl_msg", "black");
119 ((i, cc_P_IVD_1), (i, out_h_GetMsg), "to_picl_msg", "black");
120 ((i, out_h_GetMsg), (i, in_h_DataRespMsg), "to_picl_msg", "black");
121 ((i, in_h_DataRespMsg), (i, cc_P_IVD_lst), "to_picl_msg", "black");
122 ((i, cc_P_IVD_lst), (i, cc_P_V_1), "to_picl_msg", "black");
123 ((i, cc_P_V_1), (i, MResp), "to_picl_msg", "black");
124 ((i, MResp), (i, cc_P_V_lst), "to_picl_msg", "black")] /\
...
126 AssocValEqDataOfI (i, cc_P_IVD_1) i /\
127 AssocValEqCmp (i, cc_P_V_1) (i, in_h_DataRespMsg).
...
350 Axiom "ci_store_upath":
351 forall microop "i", OnCore c i => (~AccessType InitAcc i /\ ~AccessType
Evict i /\ IsAnyWrite i)=>
...
361 ExpandMacro c_case_ci_store_2 /\
```

artifact/vi_buggy_test.uarch



Intermediate output

```
113 Rule store_P_I, v:Value_1, n:Proc_3 fired.
114 prevProcs:P_I
115 prevProcReq.cl:Value_1
116 prevProcReq.vld:true
117 prevProcReq.tp:ci_store
118 Procs[Proc_3].state:P_IVD
119 Procs[Proc_3].val:Value_1
...
179 Rule receiveFromBus, n:Proc_3 fired.
180 Procs[Proc_3].state:P_V
181 Procs[Proc_3].val:Value_2
182 ackBus[Proc_3]:true
183 ----------
```

build/s3_2_reachable_set_val/_build/P_I_ci_store_0_val_eq_req_for_P_V.txt

[**Hands-On**] Inspect the evaluation result and fix the bug

Step 3: QCMBR uses property generation and model checking to produce the synthesized μSPEC$_{CCI}$ model ➔ every expression in the μSPEC$_{CCI}$ model can be traced back to a property evaluation result

Example: Take a look at the synthesized μSPEC$_{CCI}$ model to debug

```
$ cd ../../../murphi/ && vim -o artifact/vi_buggy_test.uarch
build/s3_2_reachable_set_val/_build/P_I_ci_store_0_val_eq_req_for_P_V.txt
```

QCMBR Output & CohCheck Input

Expanded/instantiated when its case_2 for store

```
110 DefineMacro "c_case_ci_store_2":
118 AddEdges[((i, CReq), (i, cc_P_IVD_1), "to_picl_msg", "black");
119 ((i, cc_P_IVD_1), (i, out_h_GetMsg), "to_picl_msg", "black");
120 ((i,                                  l_msg", "black");
121 ((i,                                  l_msg", "black");
122 ((i,                                  "black");
123 ((i, cc_P_V_1), (i, MResp), "to_picl_msg", "black");
124 ((i, MResp), (i, cc_P_V_lst), "to_picl_msg", "black")] ^
...
126 AssocValEqDataOfI (i, cc_P_IVD_1) i ^
127 AssocValEqCmp (i, cc_P_V_1) (i, in_h_DataRespMsg)
...
350 Ax
351 f
Evict
...
361 E
```

Happens-before between i visiting CReq and i visiting IV^D

The value changes when finishing transaction =>
It should have matched store's data arg =>
Property checking value-argument match during this particular transaction is cex (inside qcmbr).

Trace produced for property that checks value changes from previous state during transition from IV^D to V during store execution (inside qcmbr).

```
113 Rule store_P_I  v:Value_1, n:Proc_3 fired.
114 prevProcs:P_I
115 prevProcReq.cl Value_1
116 prevProcReq.vld:true
117 prevProcReq.tp ci_store
118 Procs[Proc_3].state:P_IVD
119 Procs[Proc_3].val:Value_1
...
179 Rule receiveFromBus, n:Proc_3 fired.
180 Procs[Proc_3].state:P_V
181 Procs[Proc_3].val:Value_2
182 ackBus[Proc_3]:true
183 ----------
```

build/s3_2_reachable_set_val/_build/P_I_ci_store_0_val_eq_req_for_P_V.txt

[**Hands-On**] Inspect the evaluation result and fix the bug

Step 3: QCMBR uses property generation and model checking to produce the synthesized $\mu SPEC_{CCI}$ model ➔ every expression in the $\mu SPEC_{CCI}$ model can be traced back to a property evaluation result

Example: Take a look at the synthesized $\mu SPEC_{CCI}$ model to debug

```
$ vim protocols/plain/mi_no_fetch_on_write_buggy.m
```

The argument for store is `v`

- `rule` defines transition relation
- invoked when a core receives store request at I state

```
287   ruleset v:Value Do
288     rule "store_P_I"
289       (p.state = P_I) & !msgBufNonEmpty & IsUndefined(gBuf.src)
290       ==>
291         get_store_req(n, v);
292         curRequsts[n].vld := true;
293         curRequsts[n].tp := store;
294         send_req (GetMsg, n, UNDEFINED, UNDEFINED);
295         p.state := P_IVD;
296         p.val := v;
297     endrule;
```

protocols/plain/mi_no_fetch_on_write_buggy.m

Define the transition when CC receives msg

```
224 Procedure ProcReceive(msg:Message; p:Proc);
...
239   case P_IVD:
240     switch msg.mtype
241     case DataRespMsg:
242       assert (msg.dst = p) "Huh not for me?";
243       pv := msg.val;
244       ps := P_V;
...
337 ruleset n:Node Do
338   rule "receiveFromBus"
...
346         -- #RECEIVING.gBuf
347         ProcReceive(gBuf, n);
348       endif;
...
```

At IV^D, receiving DataRespMsg

Value updated! Fine for load but not for store!

[Hands-On] Inspect the evaluation result and fix the bug

Step 3: QCMBR uses property generation and model checking to produce the synthesized $\mu SPEC_{CCI}$ model ➔ every expression in the $\mu SPEC_{CCI}$ model can be traced back to a property evaluation result

Example: Take a look at the synthesized $\mu SPEC_{CCI}$ model to debug

```
$ vim –o protocols/plain/mi_no_fetch_on_write_buggy.m
  protocols/plain/mi_no_fetch_on_write_coh_model.m
```

```
224 Procedure ProcReceive(msg:Message; p:Proc);
...
239   case P_IVD:
240     switch msg.mtype
241     case DataRespMsg:
242       assert (msg.dst = p) "Huh not for me?";
243       pv := msg.val;
244       ps := P_V;
...
337 ruleset n:Node Do
338   rule "receiveFromBus”
...

346         -- #RECEIVING,gBuf
347         ProcReceive(gBuf, n);
348       endif;
...
```

protocols/plain/mi_no_fetch_on_write_buggy.m

```
239   case P_IVD:
240     switch msg.mtype
241     case DataRespMsg:
242       assert (msg.dst = p) "Huh not for me?";
243       if (!isundefined(curRequsts[p].vld) &
244         curRequsts[p].vld & curRequsts[p].tp != store ) then
245         pv := msg.val;
246         -- #RETLD,CL
247       else
248         LastWrite := pv;
249       endif;
250       curRequsts[p].vld := false;
251       ps := P_V;
```

protocols/plain/mi_no_fetch_on_write_coh_model.m

[Hands-On] Inspect the evaluation result and fix the bug

Step 3: QCMBR uses property generation and model checking to produce the synthesized $\mu SPEC_{CCI}$ model ➔ every expression in the $\mu SPEC_{CCI}$ model can be traced back to a property evaluation result

Example: Take a look at the synthesized $\mu SPEC_{CCI}$ model to debug

```
$ vim –o protocols/plain/mi_no_fetch_on_write_buggy.m
  protocols/plain/mi_no_fetch_on_write_coh_model.m
```

```
224 Procedure ProcReceive(msg:Message; p:Proc);
...
239   case P_IVD:
240     switch msg.mtype
241     case DataRespMsg:
242       assert (msg.dst = p) "Huh not for me?";
243       pv := msg.val;
244       ps := P_V;
...
337 ruleset n:Node Do
338   rule "receiveFromBus”
...

346         -- #RECEIVING,gBuf
347         ProcReceive(gBuf, n);
348       endif;
...
```

protocols/plain/mi_no_fetch_on_write_buggy.m

Update cache line value only if active request is not store

```
239   case P_IVD:
240     switch msg.mtype
241     case DataRespMsg:
242       assert (msg.dst = p) "Huh not for me?";
243       if (!isundefined(curRequsts[p].vld) &
244         curRequsts[p].vld & curRequsts[p].tp != store ) then
245         pv := msg.val;
246         -- #RETLD,CL
247       else
248         LastWrite := pv;
249       endif;
250       curRequsts[p].vld := false;
251       ps := P_V;
```

proto

Otherwise if request is a store, then we treat data only as an ack and transition to V state

Roadmap: Apply QCMBR+CohCheck on a buggy VI protocol

- Step 1: [Input] Annotations
- Step 2: [Output] Run QCMBR+CohCheck on a buggy VI protocol
- Step 3: [Output] Inspect the verification results and fix the bug
- **Step 4: [Output] Re-run CohCheck on the fixed VI μSPEC$_{CCI}$ model**

[Hands-On] : Re-run CohCheck on the synthesized μSPEC for fixed VI

Step 4: Re-run QCMBR on the fixed protocol model and use CohCheck to conduct litmus-test based evaluation against the synthesized $\mu SPEC_{ccl}$ model

Example: Coherence verification on synthesized $\mu SPEC_{ccl}$ model of the fixed VI protocol introduced in previous slide

```
$ vimdiff artifact/{vi_test,vi_buggy_test}.uarch
```

```
116 DefineMacro "c_case_ci_store_2":
117 AddEdge ((i, CReq), (i, MResp), "", "black") ∧
118 AddEdges[((i, CReq), (i, cc_P_IVD_1), "to_picl_msg", "black");
119 ((i, cc_P_IVD_1), (i, out_h_GetMsg), "to_picl_msg", "black");
120 ((i, out_h_GetMsg), (i, in_h_DataRespMsg), "to_picl_msg", "black");
121 ((i, in_h_DataRespMsg), (i, cc_P_IVD_lst), "to_picl_msg", "black");
122 ((i, cc_P_IVD_lst), (i, cc_P_V_1), "to_picl_msg", "black");
123 ((i, cc_P_V_1), (i, MResp), "to_picl_msg", "black");
124 ((i, MResp), (i, cc_P_V_lst), "to_picl_msg", "black")] ∧
...
127 AssocValEqCmp (i, cc_P_V_1) (i, in_h_DataRespMsg).
```

artifact/vi_buggy_test.uarch

```
116 DefineMacro "c_case_ci_store_2":
117 AddEdge ((i, CReq), (i, MResp), "", "black") ∧
118 AddEdges[((i, CReq), (i, cc_P_IVD_1), "to_picl_msg", "black");
119 ((i, cc_P_IVD_1), (i, out_h_GetMsg), "to_picl_msg", "black");
120 ((i, out_h_GetMsg), (i, in_h_DataRespMsg), "to_picl_msg", "black");
121 ((i, in_h_DataRespMsg), (i, cc_P_IVD_lst), "to_picl_msg", "black");
122 ((i, cc_P_IVD_lst), (i, cc_P_V_1), "to_picl_msg", "black");
123 ((i, cc_P_V_1), (i, MResp), "to_picl_msg", "black");
124 ((i, MResp), (i, cc_P_V_lst), "to_picl_msg", "black")] ∧
...
127 AssocValEqDataOfI (i, cc_P_V_1) i.
```

artifact/vi_test.uarch

[Hands-On] : Re-run CohCheck on the synthesized μSPEC for fixed VI

Step 4: Re-run QCMBR on the fixed protocol model and use CohCheck to conduct litmus-test based evaluation against the synthesized $\mu SPEC_{ccl}$ model

Example: Coherence verification on synthesized $\mu SPEC_{ccl}$ model of the fixed VI protocol introduced in previous slide

```
$ cd ../check/sandbox/murphi_tests && ./eval.sh run_fix
```

est	Time	TimeSMT	Bugs	Strict	Inst#
2_2W	0.45	1.27	No	No	8
co-iriw	0.54	2.68	No	No	10
co-mp	0.20	1.19	No	No	8
coRR	0.11	0.49	No	No	7
coRW1	0.02	0.03	No	No	3
coRW2_P	0.15	0.63	No	No	7
coRW2	0.14	0.60	No	No	7
coWR_1thd	0.03	0.06	No	No	3
coWR2_2	0.09	0.33	No	No	6
coWR2	0.08	0.31	No	No	6
coWR	0.14	0.61	No	No	7
coWW_R	0.15	0.65	No	No	7
coWW	0.04	0.07	No	No	4
n4	0.71	1.64	No	No	8
n5	0.17	0.58	No	No	6
S_poss	0.30	1.10	No	No	8
ssl	0.07	0.15	No	No	4
W_evict2	0.02	0.03	No	No	3
W_evict	0.03	0.03	No	No	3
W_R_fr	0.04	0.07	No	No	4
W_R_rf	0.03	0.07	No	No	4
W_RW_rf	0.12	0.60	No	No	7

No bugs and no too-strong results

result after running eval.sh

Other outputs

Run full qcmbr to generate μSPEC$_{CCI}$

- `./full_flow_2.sh`

Run full CohCheck to verify μSPEC$_{CCI}$-MCM compliance:

- `./eval.sh mcm`