A Leap Forward in Architectural Formal Convergence Using Automated Case Splitting

Dr. Shahid Ikram
Distinguished Engineer
Marvell

Erik Seligman
Senior Product Engineering Architect
Cadence

Agenda

- Protocol Verification
- Architectural Modeling
- The Closure Challenge
- Proof Automation
- Case Splitting
- Results and Conclusions

Protocol Verification

JULY 9–13, 2023

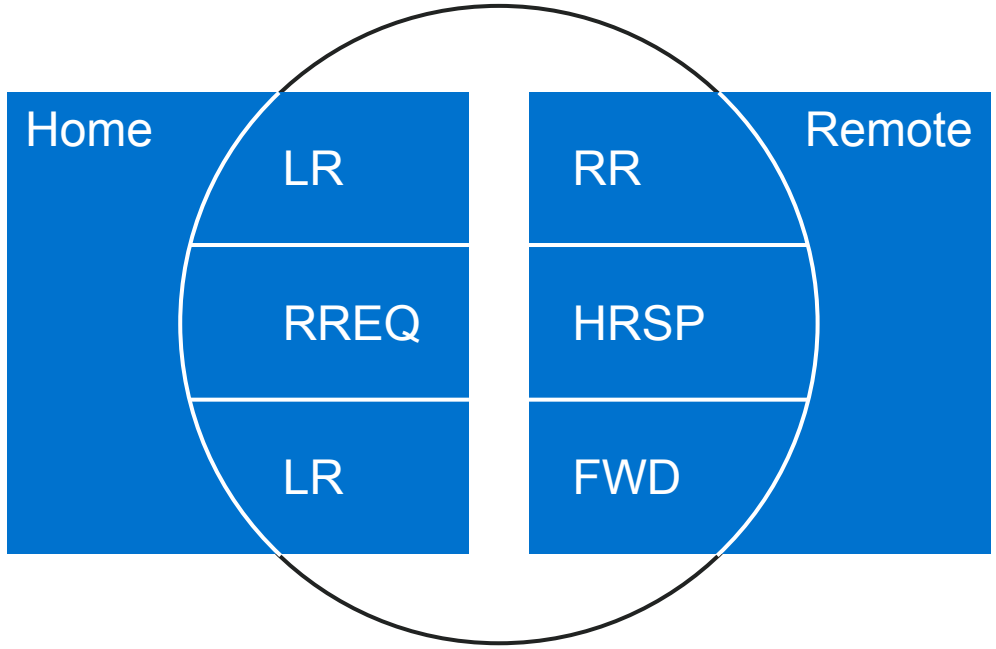
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Agents, roles, tables

Agents
Concurrent interacting state machines

Roles
Abstract states

Tables
Transitions



ASCII tables

Remote table(s) for a node						
Current state		Next state		Outputs		Inputs
CMD	**ST**	**CMD**	**ST**	**H**	**N2**	**N3**	**REQ/RESP/FRWD**
none	I	OCI_RD	I	OCI_RD (Read from remote node – i.e. home)	-	-	LCL_RD_RMT_A (local read – remote address)
OCI_RD	I	none	S	-	-	-	RD_RSP (read response)
none	S	none	I	INV_RSP (Invalidate Response)	-	-	INV (Invalidate)

Home table(s)												
Current state					Next state					Outputs			Inputs
CMD	**H**	**N1**	**N2**	**N3**	**CMD**	**H**	**N1**	**N2**	**N3**	**N1**	**N2**	**N3**	**REQ/RESP/FRWD**
none	E	I	I	I	none	S	S	I	I	RD_RSP	-	-	OCI_RD (Read from remote node)
none	S	S	I	I	LCL_WR	S	**S->I**	I	I	INVAL	-	-	LCL_WR_LCL_A (local write to home address)
LCL_WR	S	S->I	I	I	none	M	I	I	I	-	-	-	INV_RSP (Invalidate response)

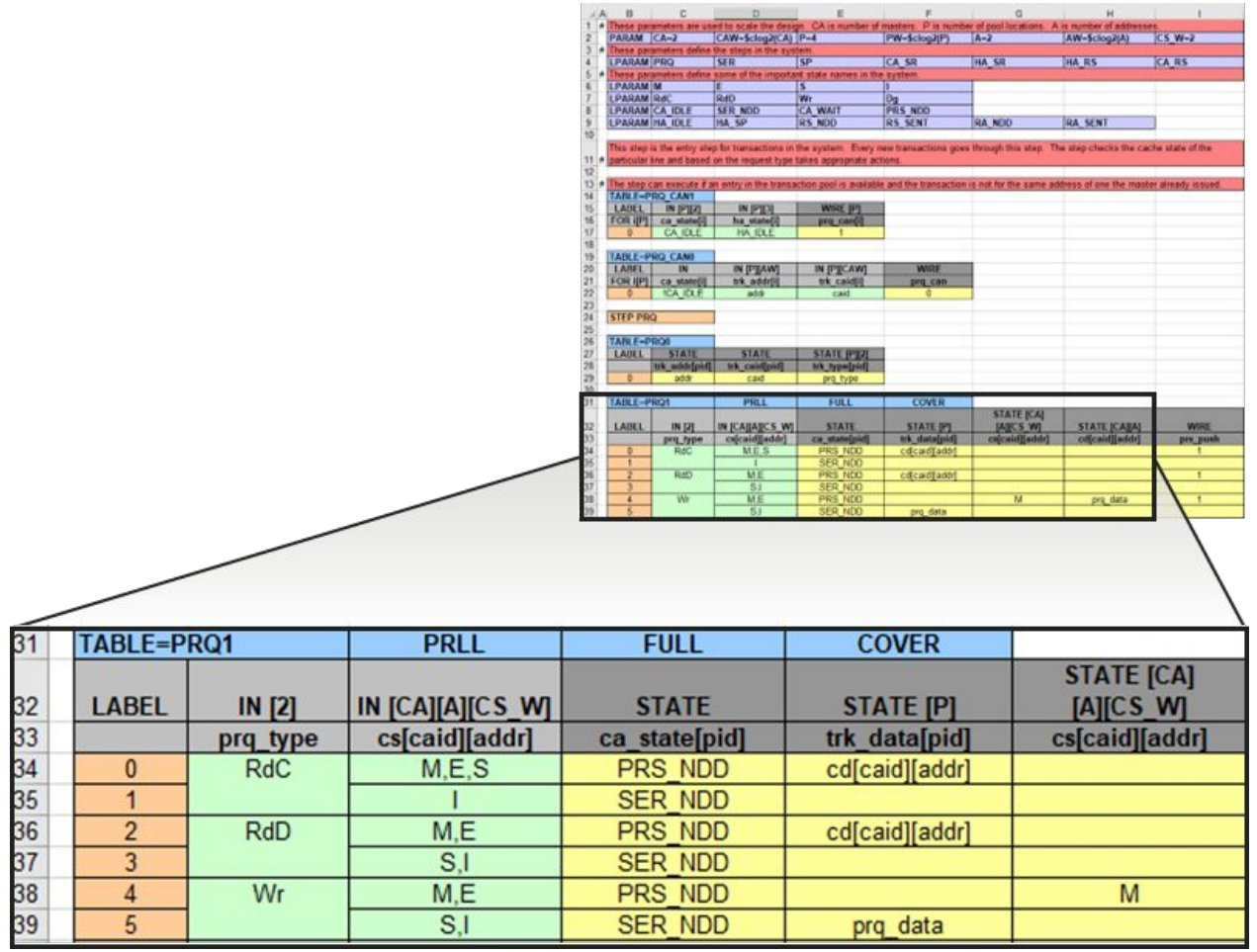
Architectural Modeling

JULY 9–13, 2023

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Architectural modeling tool

- A set of tables
- Each table models a set of transitions
- An agent's behavior can be modelled through a set of tables
- Agents' interaction is modeling using data-structures called channels



31	TABLE=PRQ1		PRLL	FULL	COVER	
32	LABEL	IN [2]	IN [CA][A][CS_W]	STATE	STATE [P]	STATE [CA][A][CS_W]
33		prq_type	cs[caid][addr]	ca_state[pid]	trk_data[pid]	cs[caid][addr]
34	0	RdC	M,E,S	PRS_NDD	cd[caid][addr]	
35	1		I	SER_NDD		
36	2	RdD	M,E	PRS_NDD	cd[caid][addr]	
37	3		S,I	SER_NDD		
38	4	Wr	M,E	PRS_NDD		M
39	5		S,I	SER_NDD	prq_data	

Protocol checks

- Protocol checks
  - PRLL
    - No two transitions have same trigger
  - Full
    - No missing transitions
  - Cover
    - All transitions are reachable
- End-to-end checks
  - Deadlock
  - Cache coherence
  - Data integrity
  - Good state

	TABLE=PRQ1		PRLL	FULL	COVER	
	LABEL	IN [2]	IN [CA][A][CS_W]	STATE	STATE [P]	STATE [CA][A][CS_W]
		prq_type	cs[caid][addr]	ca_state[pid]	trk_data[pid]	cs[caid][addr]
34	0	RdC	M,E,S	PRS_NDD	cd[caid][addr]	
35	1		I	SER_NDD		
36	2	RdD	M,E	PRS_NDD	cd[caid][addr]	
37	3		S,I	SER_NDD		
38	4	Wr	M,E	PRS_NDD		M
39	5		S,I	SER_NDD	prq_data	

#	Targeted properties - data integrity and deadlock freedom						
	TABLE=DATA_INTEGRITY	NEVER					
	LABEL	IN					
		check					
	0	ERROR					
	TABLE=ACTIVE						
	LABEL	IN	IN	WIRE			
	FOR i[P]	ca_state [i]	ha_state [i]	active			
	0	OR !CA_IDLE	OR !HA_IDLE	1			
	TABLE=DEADLOCK			NEVER			
	LABEL	IN	IN	IN	IN	IN	IN
		active	ser_can	sp_can	ca_sr_can	ha_sr_can	ha_rs_can
	0	1	0	0	0	0	0

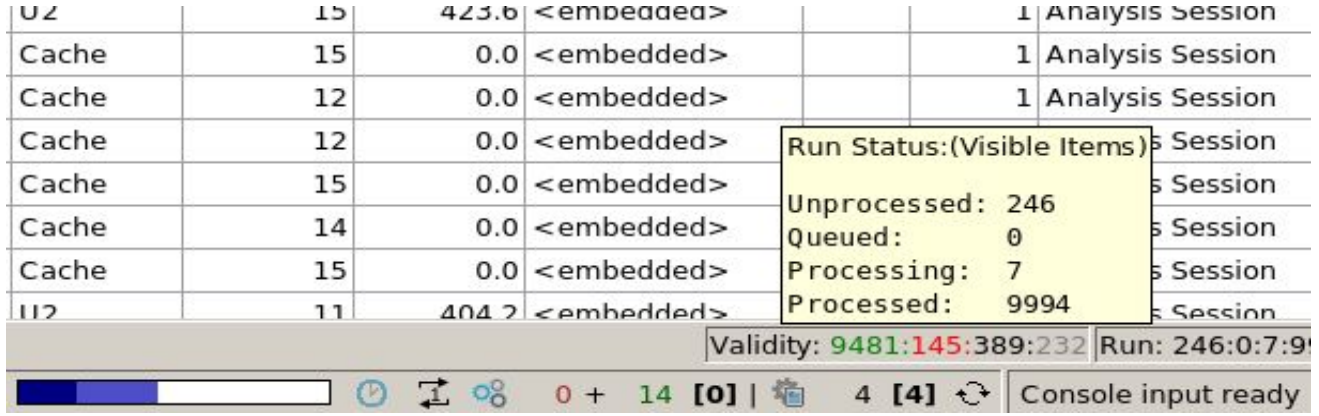
The Closure Challenge

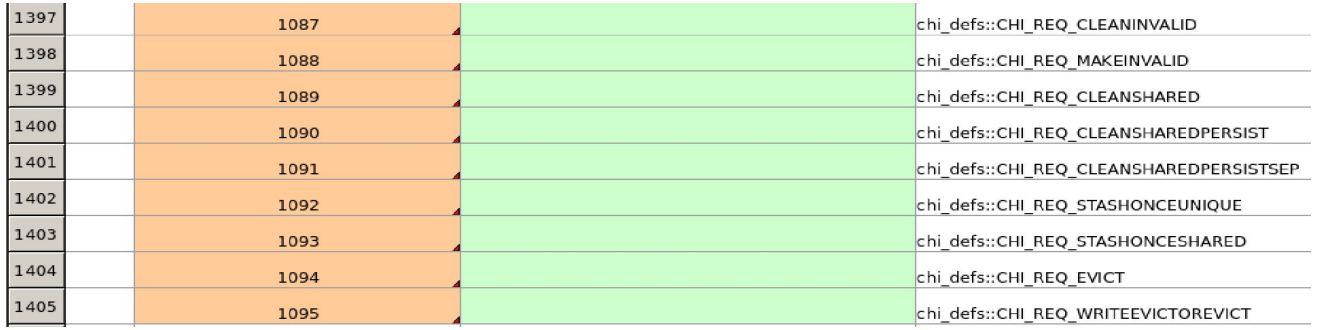
JULY 9–13, 2023

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

CHI-E modeling and verification

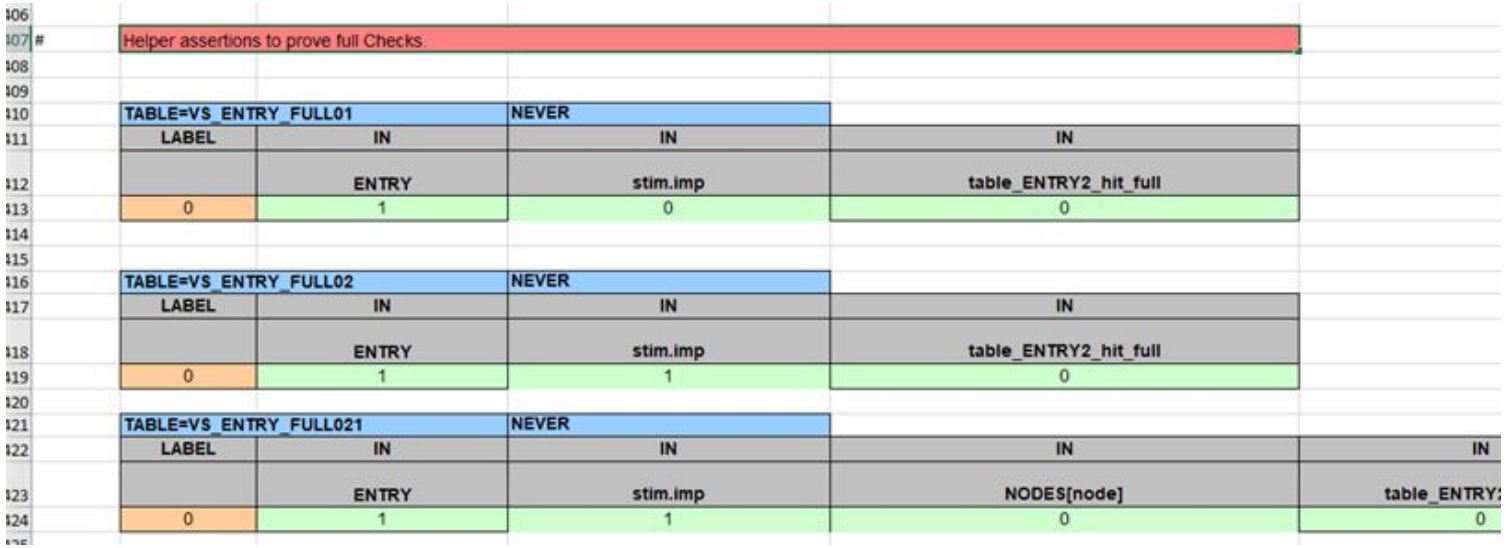
- 8 major tables
  - **PRLL** are easy to prove
  - **Cover** using IVA
  - **Full** is **hard to prove**
- 40+ supporting tables
- 5000 transitions
- 10000+ properties
  - Covers, parallel and full checks
- 50+ end-to-end properties
  - Cache coherence, data, deadlock
  - **All hard to prove**





Manual assume-guarantee for FULL

- Helper assertions
- Dividing a table
- Sub-tables Full checks
  - Table_ENTRY2_hit_full is an auto-generated signal to check if an entry in the table is hit or not
- 50+/- tables to model helper assertions for this table of 150 entries
  - Imagine the maintenance effort



406					
407	#	Helper assertions to prove full Checks.			
408					
409					
410		TABLE=VS_ENTRY_FULL01	NEVER		
411		LABEL	IN	IN	IN
412			ENTRY	stim.imp	table_ENTRY2_hit_full
413		0	1	0	0
414					
415					
416		TABLE=VS_ENTRY_FULL02	NEVER		
417		LABEL	IN	IN	IN
418			ENTRY	stim.imp	table_ENTRY2_hit_full
419		0	1	1	0
420					
421		TABLE=VS_ENTRY_FULL021	NEVER		
422		LABEL	IN	IN	IN
423			ENTRY	stim.imp	NODES[node]
424		0	1	1	0

Challenges

10,000 properties

Manual effort

Creative helper assertions
Prove
Convert helper assertions to assumptions in top-level proofs
Completeness for the split cases

Maintenance

Pages and pages of scripts
Debugging the scripts – Bugs induced are proportional to the number of lines written

Safety

No automatic completeness criteria
No safety net
Missing theorem prover like HOL

Proof Automations (Proof Structure)

JULY 9–13, 2023

MOSCONE WEST CENTER

SAN FRANCISCO, CA, USA

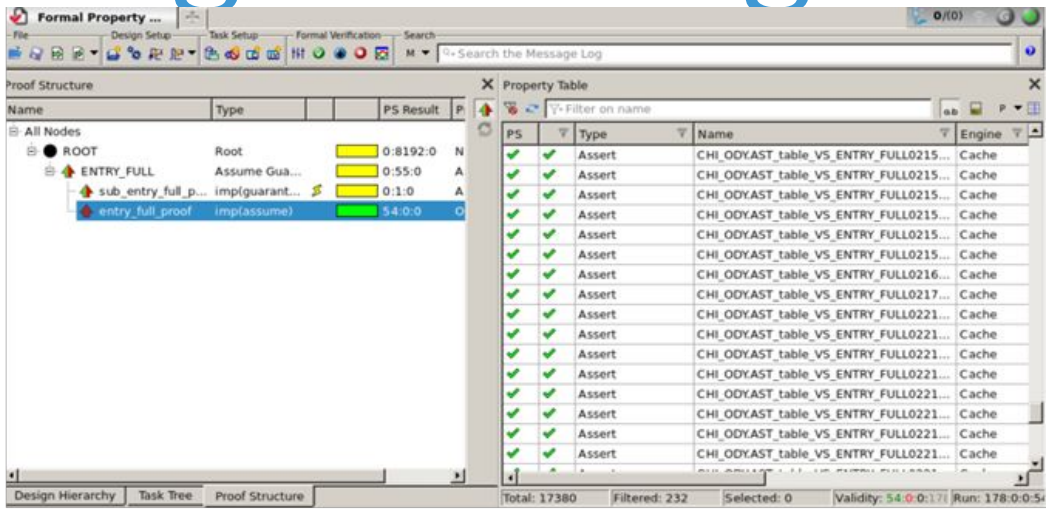
Assume-guarantee for FULL

- Helper assertions
- Proved separately
- Used as an assumption in the guarantee part

406					
407	#	Helper assertions to prove full Checks.			
408					
409					
410		TABLE=VS_ENTRY_FULL01	NEVER		
411		LABEL	IN	IN	IN
412			ENTRY	stim.imp	table_ENTRY2_hit_full
413		0	1	0	0
414					
415					
416		TABLE=VS_ENTRY_FULL02	NEVER		
417		LABEL	IN	IN	IN
418			ENTRY	stim.imp	table_ENTRY2_hit_full
419		0	1	1	0
420					
421		TABLE=VS_ENTRY_FULL021	NEVER		
422		LABEL	IN	IN	IN
423			ENTRY	stim.imp	NODES[node]
424		0	1	1	0

Automated proof using assume-guarantee

- Advantages:
  - Proof management
  - Proving helper assertions and then converting them to assumptions is fully automated
- Pitfalls:
  - Still need to create all the helper assertions tables manually.
    Not a tool's fault




```
# Set up Clocks and Resets
clock clk
reset rst


set_prove_orchestration on

set_prove_time_limit 7200

#Create root node
proof_structure -init ROOT -copy_all

# A branch for ENTRY table FULL check proof
proof_structure -create assume_guarantee -op_name ENTRY_FULL
-imp_name {sub_entry_full_proofs entry_full_proof}
-from ROOT -property { {ROOT::CHI_ODY.AST_table_ENTRY2_full} {ROOT::CHI_ODY.AST_table_VS_ENTRY_FULL*}}

# Run two task
prove -bg -task {entry_full_proof}
prove -bg -task {sub_entry_full_proofs}
```

Case splitting

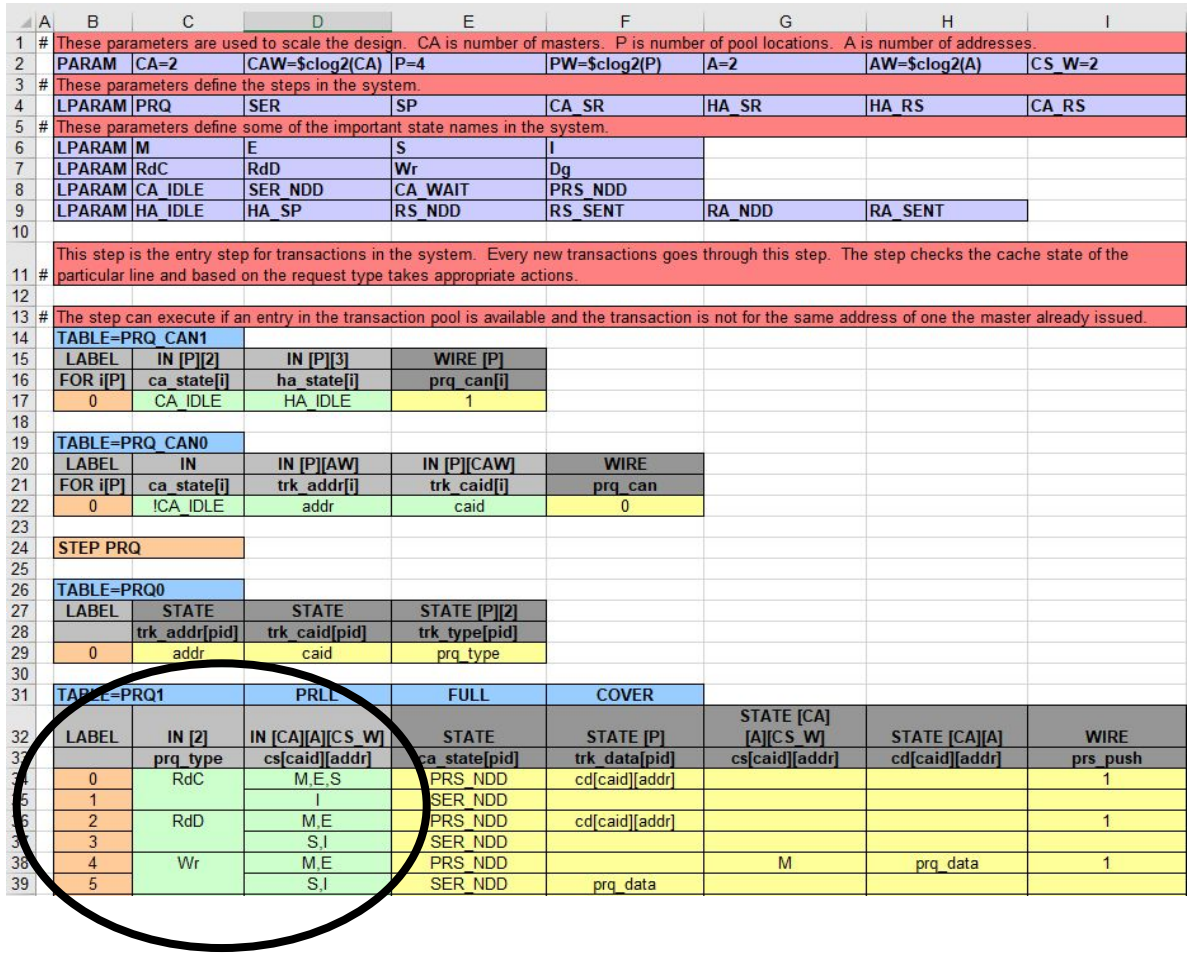
- Proof by cases:

  A mathematical proof by exhausting all the exclusive possibilities.

  Theorem to prove Alpha:

  ((a || ~a) → Alpha) == Alpha

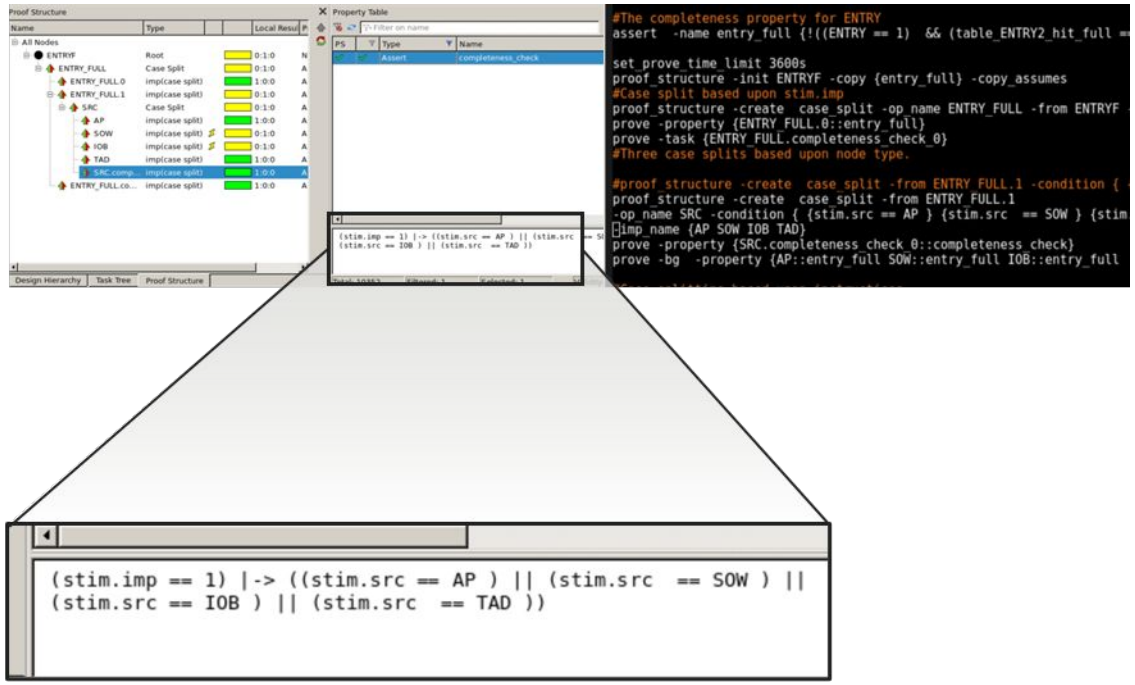
  ((a || ~a) → Alpha) ==
  ((a) → Alpha) && ((~a) → Alpha)



	A	B	C	D	E	F	G	H	I
1	#	These parameters are used to scale the design. CA is number of masters. P is number of pool locations. A is number of addresses.							
2		PARAM	CA=2	CAW=$clog2(CA)	P=4	PW=$clog2(P)	A=2	AW=$clog2(A)	CS_W=2
3	#	These parameters define the steps in the system.							
4		LPARAM	PRQ	SER	SP	CA_SR	HA_SR	HA_RS	CA_RS
5	#	These parameters define some of the important state names in the system.							
6		LPARAM	M	E	S	I			
7		LPARAM	RdC	RdD	Wr	Dg			
8		LPARAM	CA_IDLE	SER_NDD	CA_WAIT	PRS_NDD			
9		LPARAM	HA_IDLE	HA_SP	RS_NDD	RS_SENT	RA_NDD	RA_SENT	
10									
11	#	This step is the entry step for transactions in the system. Every new transactions goes through this step. The step checks the cache state of the particular line and based on the request type takes appropriate actions.							
12									
13	#	The step can execute if an entry in the transaction pool is available and the transaction is not for the same address of one the master already issued.							
14		TABLE=PRQ_CAN1							
15		LABEL	IN [P][2]	IN [P][3]	WIRE [P]				
16		FOR i[P]	ca_state[i]	ha_state[i]	prq_can[i]				
17		0	CA_IDLE	HA_IDLE	1				
18									
19		TABLE=PRQ_CAN0							
20		LABEL	IN	IN [P][AW]	IN [P][CAW]	WIRE			
21		FOR i[P]	ca_state[i]	trk_addr[i]	trk_caid[i]	prq_can			
22		0	!CA_IDLE	addr	caid	0			
23									
24		STEP PRQ							
25									
26		TABLE=PRQ0							
27		LABEL	STATE	STATE	STATE [P][2]				
28			trk_addr[pid]	trk_caid[pid]	trk_type[pid]				
29		0	addr	caid	prq_type				
30									
31		TABLE=PRQ1		PRLL	FULL	COVER			
32		LABEL	IN [2]	IN [CA][A][CS_W]	STATE	STATE [P]	STATE [CA][A][CS_W]	STATE [CA][A]	WIRE
33			prq_type	cs[caid][addr]	ca_state[pid]	trk_data[pid]	cs[caid][addr]	cd[caid][addr]	prs_push
34		0	RdC	M,E,S	PRS_NDD	cd[caid][addr]			1
35		1		I	SER_NDD				
36		2	RdD	M,E	PRS_NDD	cd[caid][addr]			1
37		3		S,I	SER_NDD				
38		4	Wr	M,E	PRS_NDD		M	prq_data	1
39		5		S,I	SER_NDD	prq_data			

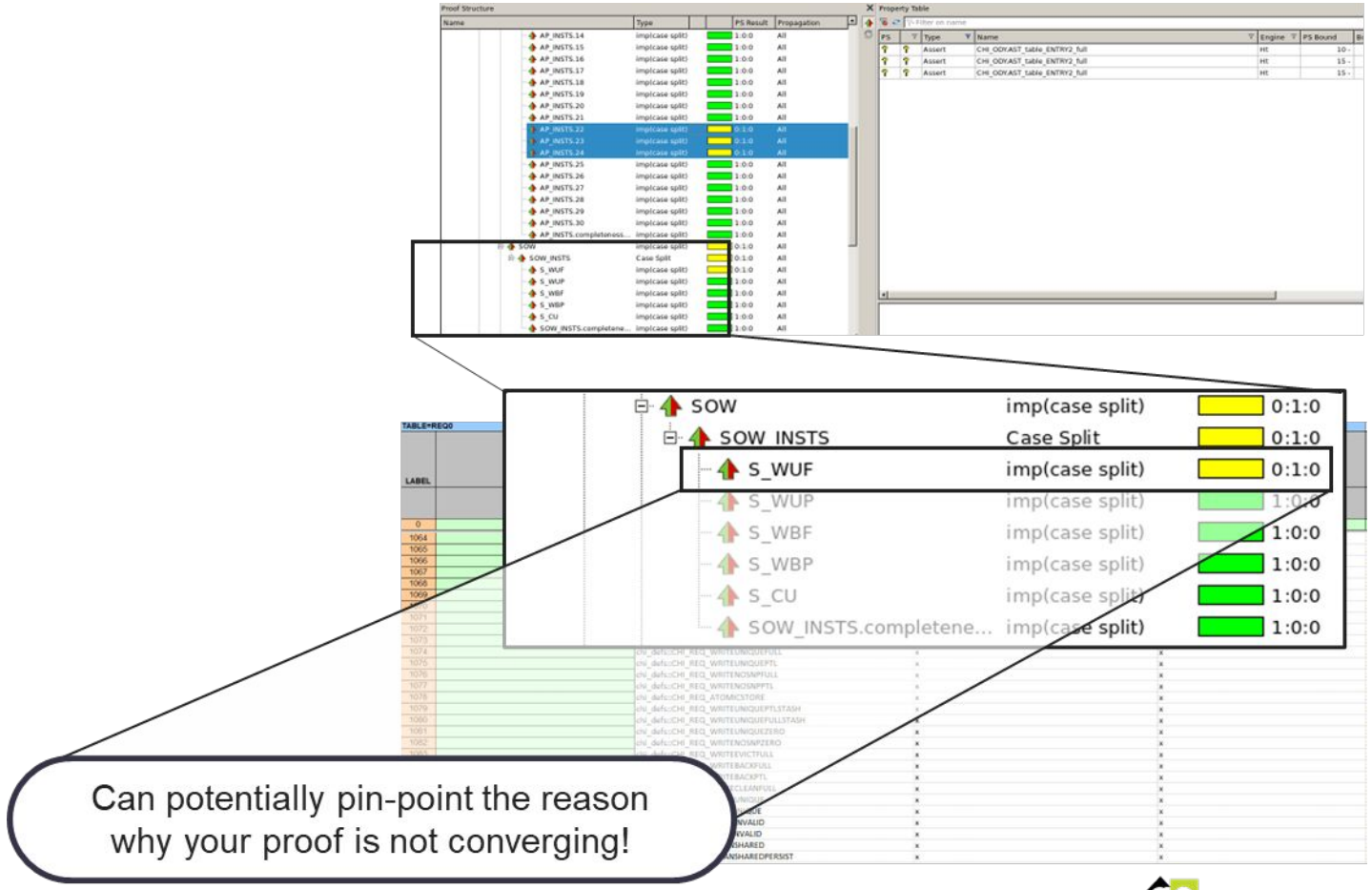
Dividing & proving FULL case through case splitting

- Fully programable
- **Completeness check ensure no missing cases**
- No need of helping assertions
- No arbitrary assumptions can be added to the sub-proofs
- Almost a theorem prover like



Using split-case analysis iteratively

One can keep splitting the problem to exactly find the reason proofs are not converging!



Results and conclusions

Verification is a risk-management job

No reasonable size design can be fully verified
All we can do is to reduce risk
to a reasonable level

Proof Automation helped us to:

Converge faster on proofs or hit deeper bounds
Weeks of effort saved using split cases(50%)
Manage proofs and TCL files better
Find bugs in TCL files when we missed cases
Enforces soundness

There is no replacement of our insight of

Design behavior
Verification methods

But then tools like "proof structure" helps just focus on that