



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

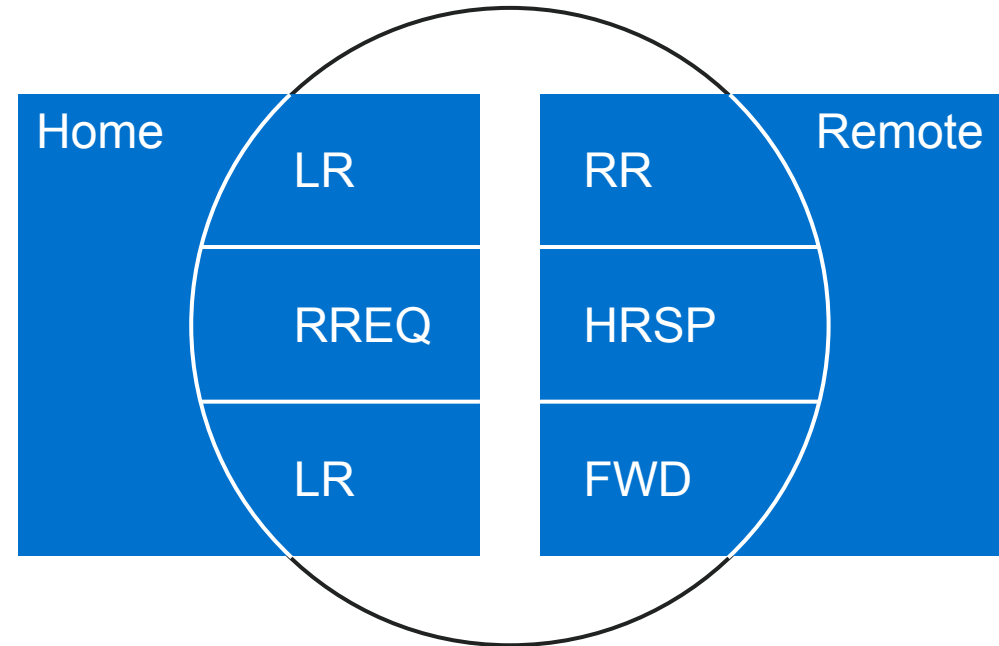
Concurrent interacting
state machines

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

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=5log2(CA)	P=4	PW=5log2(P)	A=2	AW=5log2(A)	CS_W=2
3	These parameters define the steps in the system.							
4	LPARAM	PROQ	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				
7	LPARAM	RdC	RdD	Wr	Qg			
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	This step is the entry step for transactions in the system. Every new transaction goes through this step. The step checks the cache state of the particular line and based on the request type takes appropriate actions.							
11	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.							
12								
13								
14	TABLE=PROQ CANI							
15	LABEL	IN [P][Z]	IN [P][Z]	WRITE [P]				
16	FOR [P]	ca_state[0]	ha_state[0]	prq_data[0]				
17	0	CA_IDLE	HA_IDLE					
18								
19	TABLE=PROQ CANI							
20	LABEL	IN	IN [P][AW]	IN [P][AW]	WRITE			
21	FOR [P]	ca_state[0]	ha_state[0]	prq_data[0]				
22	0	CA_IDLE						
23								
24	STEP PROQ							
25								
26								
27	TABLE=PROQ							
28	LABEL	STATE	STATE	STATE [P][Z]				
29	0	caid	caid	prq_type				
30								
31	TABLE=PROQ							
32	LABEL	IN [Z]	IN [CA][A][CS_W]	STATE	STATE [P]	STATE [CA]	STATE [CA]	WRITE
33		prq_type	cs[caid][addr]	ca_state[pid]	trk_data[pid]	cs[caid][addr]	cs[caid][addr]	prq_data
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			

31	TABLE=PRQ1	PRLL	FULL	COVER	STATE [CA]
32	LABEL	IN [Z]	IN [CA][A][CS_W]	STATE	[A][CS_W]
33		prq_type	cs[caid][addr]	ca_state[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

30						
31	TABLE=PRQ1	PRLL	FULL	COVER		
32	LABEL	IN [2]	IN [CA][A][CS_W]	STATE	STATE [P]	STATE [CA]
33		prq_type	cs[caid][addr]	ca_state[pid]	trk_data[pid]	[A][CS_W]
34	0	RdC	M,E,S	PRS_NDD	cd[caid][addr]	cs[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 ICA_IDLE	OR IHA_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
 - PRL 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

U2	15	423.6	<embedded>	1	Analysis Session
Cache	15	0.0	<embedded>	1	Analysis Session
Cache	12	0.0	<embedded>	1	Analysis Session
Cache	12	0.0	<embedded>		Session
Cache	15	0.0	<embedded>		Session
Cache	14	0.0	<embedded>		Session
Cache	15	0.0	<embedded>		Session
U2	11	404.2	<embedded>		Session

Run Status:(Visible Items)

Unprocessed: 246

Queued: 0

Processing: 7

Processed: 9994

Validity: 9481:145:389:232 | Run: 246:0:7:9

0 + 14 [0] | 4 [4]

Console input ready

1397	1087	chi_defs::CHI_REQ_CLEANINVALID
1398	1088	chi_defs::CHI_REQ_MAKEINVALID
1399	1089	chi_defs::CHI_REQ_CLEANSHARED
1400	1090	chi_defs::CHI_REQ_CLEANSHAREDPERERSIST
1401	1091	chi_defs::CHI_REQ_CLEANSHAREDPERERSISTSEP
1402	1092	chi_defs::CHI_REQ_STASHONCEUNIQUE
1403	1093	chi_defs::CHI_REQ_STASHONCESHARED
1404	1094	chi_defs::CHI_REQ_EVICT
1405	1095	chi_defs::CHI_REQ_WRITEEVICTIONEVICT



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

#106					
#107	Helper assertions to prove full Checks.				
#108					
#109					
#110	TABLE=VS_ENTRY_FULL01		NEVER		
#111	LABEL	IN	IN	IN	
#112		ENTRY	stim.imp	table_ENTRY2_hit_full	
#113	0	1	0	0	
#114					
#115	TABLE=VS_ENTRY_FULL02		NEVER		
#116	LABEL	IN	IN	IN	
#117		ENTRY	stim.imp	table_ENTRY2_hit_full	
#118	0	1	1	0	
#119					
#120	TABLE=VS_ENTRY_FULL021		NEVER		
#121	LABEL	IN	IN	IN	IN
#122		ENTRY	stim.imp	NODES[node]	table_ENTRY:
#123	0	1	1	0	0
#124					



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

106					
107	#	Helper assertions to prove full Checks.			
108					
109					
110		TABLE=VS_ENTRY_FULL01		NEVER	
111		LABEL	IN	IN	IN
112			ENTRY	stim.imp	table_ENTRY2_hit_full
113		0	1	0	0
114					
115					
116		TABLE=VS_ENTRY_FULL02		NEVER	
117		LABEL	IN	IN	IN
118			ENTRY	stim.imp	table_ENTRY2_hit_full
119		0	1	1	0
120					
121		TABLE=VS_ENTRY_FULL021		NEVER	
122		LABEL	IN	IN	IN
123			ENTRY	stim.imp	NODES[node]
124		0	1	1	0
125					table_ENTRY:



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

The screenshot shows the 'Formal Property' tool interface. On the left, the 'Proof Structure' tree shows a hierarchy: ROOT (Root) -> ENTRY_FULL (Assume Gua...) -> sub_entry_full_p... (imp(guarant...)) -> entry_full_proof (imp(assume)). The 'entry_full_proof' node is highlighted. On the right, the 'Property Table' is displayed, showing a list of properties with columns for PS, Type, Name, and Engine. The table contains 17 rows of properties, all of which are 'Assert' type and have a 'Cache' engine. The status bar at the bottom indicates 'Total: 17380', 'Filtered: 232', 'Selected: 0', 'Validity: 54:0:17', and 'Run: 178:0:0:54'.

PS	Type	Name	Engine
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache
✓	Assert	CHI_ODYAST_table_VS_ENTRY_FULL0215...	Cache

```
# 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:

$$((a \parallel \sim a) \rightarrow \text{Alpha}) == \text{Alpha}$$

$$((a \parallel \sim a) \rightarrow \text{Alpha}) ==$$

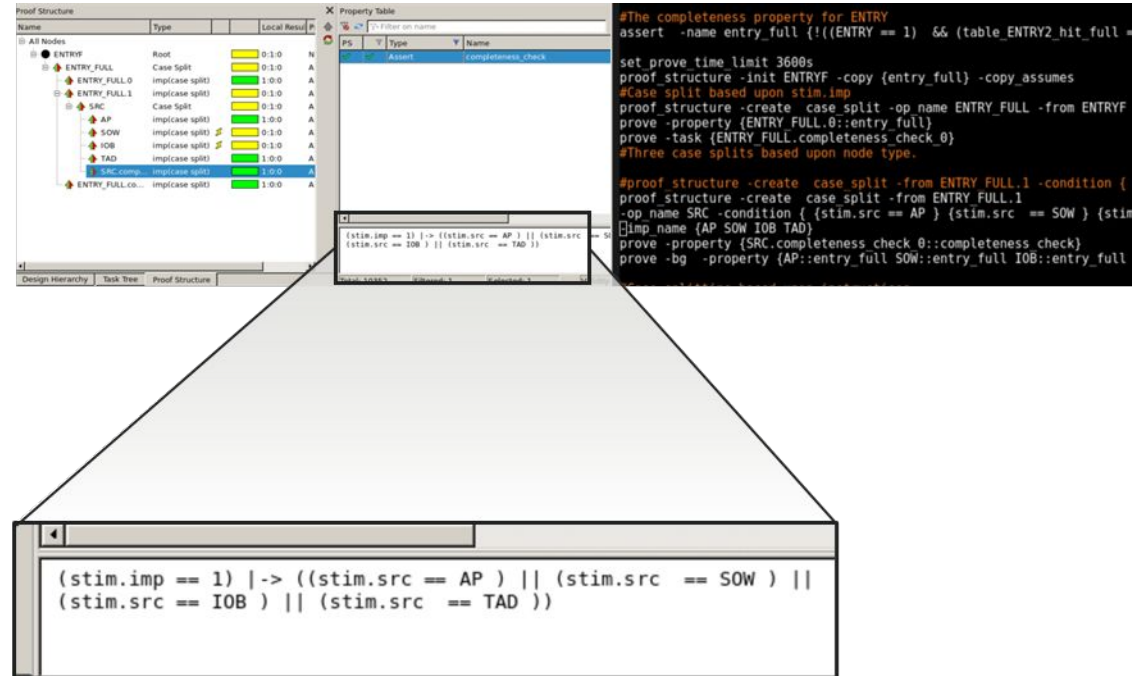
$$((a \rightarrow \text{Alpha}) \&\& ((\sim a) \rightarrow \text{Alpha}))$$

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	ICA_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]	STATE [CA][A]	WIRE
33		prq_type	cs[caid][addr]	ca_state[pid]	trk_data[pid]	A][CS_W]	cd[caid][addr]	prs_push
34		0	M,E,S	PRS_NDD	cd[caid][addr]			1
35		1	I	SER_NDD				
36		2	M,E	PRS_NDD	cd[caid][addr]			1
37		3	S,I	SER_NDD				
38		4	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!

The screenshot displays two windows from a formal verification tool. The 'Proof structure' window on the left shows a list of proof goals (AP_INVSTs) and their status. The 'Property Table' window on the right shows a table with columns 'PS', 'V', 'Type', 'Name', 'Engine', and 'PS Bound'.

PS	V	Type	Name	Engine	PS Bound
Assert			CH_OOKAST_tdata_ENTRV2_full	HE	10
Assert			CH_OOKAST_tdata_ENTRV2_full	HE	10
Assert			CH_OOKAST_tdata_ENTRV2_full	HT	15

TABLE=REQ0

LABEL

0

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1445

1446

1447

1448

1449

1450

1451

1452

1453

1454

1455

1456

1457

Can potentially pin-point the reason why your proof is 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**

