Chasing Minimal Inductive Validity Cores in Hardware Model Checking

Ryan Berryhill
Andreas Veneris
Outline

• Motivation
• Background
• The UMIVC Algorithm
• Experiments
• Conclusion
• **Motivation**
• Background
• The UMIVC Algorithm
• Experiments
• Conclusion
Motivation

• Model checkers give limited feedback for passing instances
  – Failing instance: counter-example showing how to falsify the property
  – Passing: a safe inductive invariant (for safety properties)

• A similar situation in SAT solving
  – SAT: satisfying assignment, explains how to satisfy the formula
  – UNSAT: resolution refutation? RUP proof?
  – But UNSAT cores provide usable feedback

• Inductive Validity Cores (IVCs) [1]: like UNSAT cores for safety checking
  – Originally developed for software safety checking
  – This paper introduces related algorithms for the hardware context and a wide range of variants

Outline

• Motivation
• **Background**
• The UMIVC Algorithm
• Experiments
• Conclusion
Safety Checking

• Given a safety checking problem \((\text{Init}, \text{Tr}, \text{Bad})\)

• **UNSAFE iff** there is a path from an initial state to a bad state:
  – A counter-example serves as a certificate
  – Typically something a human user can understand

• **SAFE iff** there exists a safe inductive invariant
  – IC3 returns safe inductive invariants in CNF
  – Not related to the given input, just an opaque artifact of verification
  – Not likely to be understood by a human
Inductive Validity Cores

• Even in (rare) cases where a simple, understandable safe inductive invariant exists, we may not find it

• Inductive Validity Cores (IVCs)
  – An abstraction of the circuit that is itself SAFE for the given property

• Minimal IVCs (MIVCs)
  – An IVC where no abstraction is also an IVC

• IVCs relate directly to the user’s input and are fit for human consumption
Inductive Validity Cores

Init = \(\overline{v_1}\)

Bad = \(v_1\)
Inductive Validity Cores

\[ \text{Init} = (\overline{v_1}) \]
\[ \text{Bad} = (v_1) \]
Inductive Validity Cores

- Abstraction operation: replace the output of any removed gates with a new primary input
- This abstraction is an IVC
• This is an MIVC: removing any gate would make it unsafe
Inductive Validity Cores

- This is an unsafe abstraction
- Adding any gate would make it safe, so it is maximal (an MUA)
Finding a Single MIVC

• The IVC_UCBF algorithm [1]

• Step 1: IVC_UC
  – Given a safe inductive invariant \( Inv \), find a high-level UNSAT core of the formula \( Inv \land Tr \land \neg Inv' \)
  – Minimization is only done over \( Tr \), using one clause group per gate
  – Yields a hopefully-small but non-minimal IVC

• Step 2: IVC_BF
  – Repeatedly remove a gate and check for safety
  – If UNSAFE, back out the removal of the gate
  – Yields an MIVC

Finding All MIVCs

• A MARCO-based algorithm [1], simplified here:

• A CNF formula called the map tracks which abstractions are explored

• Pick an arbitrary seed (unexplored abstraction)

• Is it SAFE? Use IVC_UCBF to shrink it to an MIVC
  – Block all supersets by adding a clause to the map

• Is it UNSAFE? Use brute-force to grow it to an MUA
  – Block all subsets by adding a clause to the map

Finding All MIVCs

• A MARCO-based algorithm [1], simplified here:

• A CNF formula called the map tracks which abstractions are explored

• Pick an arbitrary seed (unexplored abstraction)

• Is it SAFE? Use IVC_UCBF to shrink it to an MIVC
  – Block all supersets by adding a clause to the map

• Is it UNSAFE? Use brute-force to grow it to an MUA
  – Block all subsets by adding a clause to the map

Finding All MIVCs

• A MARCO-based algorithm [1], simplified here:

• A CNF formula called the *map* tracks which abstractions are explored

• Pick an arbitrary *seed* (unexplored abstraction)

• Is it SAFE? Use IVC_UCBF to shrink it to an MIVC
  – Block all supersets by adding a clause to the map

• Is it UNSAFE? Use brute-force to *grow* it to an MUA
  – Block all subsets by adding a clause to the map

Finding All MIVCs

• A MARCO-based algorithm [1], simplified here:

• A CNF formula called the map tracks which abstractions are explored

  maximum-cardinality

• Pick an arbitrary seed (unexplored abstraction)

• Is it SSAFE? Use IVC_UCBF to shrink it to an MIVC
  – Block all supersets by adding a clause to the map

  Use IVC_UC, can’t find MIVCs until termination

  No need to grow because the seed is already maximal

• Is it UNSAFE? Use brute force to grow it to an MUA
  – Block all subsets by adding a clause to the map

Finding All MIVCs

Algorithm 1 MARCO for MIVC extraction

Input: safety checking problem \((\text{Init}, \text{Tr}, \text{Bad})\)
Output: set of all MIVCs of \((\text{Init}, \text{Tr}, \text{Bad})\)

1: \(\text{MIVCs} \leftarrow \emptyset\)
2: \(\text{map} \leftarrow \top\)
3: while \(\text{map}\) is SAT do
4: \(\text{seed} \leftarrow \text{getUnexplored} (\text{map})\)
5: if \((\text{Init}, \text{seed}, \text{Bad})\) is SAFE then
6: \(\text{mivc} \leftarrow \text{shrink} (\text{seed})\)
7: \(\text{MIVCs} \leftarrow \text{MIVCs} \cup \{\text{mivc}\}\)
8: \(\text{map} \leftarrow \text{map} \land \text{blockUp} (\text{mivc})\)
9: else
10: \(\text{mua} \leftarrow \text{grow} (\text{seed})\)
11: \(\text{map} \leftarrow \text{map} \land \text{blockDown} (\text{mua})\)
12: end if
13: end while
14: return \(\text{MIVCs}\)
Finding All MIVCs

**Algorithm 1** MARCO for MIVC extraction

**Input:** safety checking problem \((Init, Tr, Bad)\)

**Output:** set of all MIVCs of \((Init, Tr, Bad)\)

```plaintext
while Unexplored seeds exist do
    seed ← getUnexplored()
    if seed is SAFE then
        shrink
    else
        grow
        blockDown
    end if
end while
```
Finding All MIVCs

Algorithm 1 MARCO for MIVC extraction

Input: safety checking problem \((\text{Init}, \text{Tr}, \text{Bad})\)

Output: set of all MIVCs of \((\text{Init}, \text{Tr}, \text{Bad})\)

1: \(MIVCs \leftarrow \emptyset\)
2: \(map \leftarrow \top\)

Seed Extraction Loop

14: return \(MIVCs\)
Outline

• Motivation
• Background
• The UMIVC Algorithm
• Experiments
• Conclusion
• CAMUS is another well-known MUS enumeration algorithm

• Minimal correction subset (MCS): if you remove this set of clauses from the formula, the result is SAT
  – Hitting set duality: a minimal hitting set of the MCSes is an MUS
  – Similar definitions extend to circuits

• Find all MCSes, then find MUSes/MIVCs as hitting sets

• Simpler explanation
  – Find and block everything that is UNSAFE/SAT
  – Minimal unexplored seeds are now MIVCs/MUSes
Algorithm 2  \textsc{Camus} for MIVC extraction

\textbf{Input:} safety checking problem \((Init, Tr, Bad)\)
\textbf{Output:} set of all MIVCs of \((Init, Tr, Bad)\)

1: \(MIVCs \leftarrow \emptyset\)
2: \(map \leftarrow \top\)
3: \textbf{while} more MCSes exist \textbf{do}
4: \hspace{1em} \(mcs \leftarrow \text{FindMCS}(Init, Tr, Bad)\)
5: \hspace{1em} \(map \leftarrow map \land \text{blockDown}(Tr \setminus mcs)\)
6: \textbf{end while}
7: \textbf{while} map is SAT \textbf{do}
8: \hspace{1em} \(mivc \leftarrow \text{getUnexploredMin}(map)\)
9: \hspace{1em} \(MIVCs \leftarrow MIVCs \cup \{mivc\}\)
10: \hspace{1em} \(map \leftarrow map \land \text{blockUp}(mivc)\)
11: \textbf{end while}
12: \textbf{return} \(MIVCs\)
Algorithm 2 \textsc{Camus} for MIVC extraction

\textbf{Input}: safety checking problem \((Init, Tr, Bad)\)

\textbf{Output}: set of all MIVCs of \((Init, Tr, Bad)\)

1: 

2: 

3: \textbf{while} More unsafe abstractions exist 

4: \textbf{end while}

5: \textbf{while} Unexplored seeds exist 

6: \textbf{end while}

7: \textbf{while} 

8: \textbf{end while}

9: 

10: 

11: 

12: 

\textbf{Find MUA}

\textbf{blockDown}

\textbf{getBlockMin}

\textbf{blockUp}
Algorithm 2 \textsc{camus} for MIVC extraction

\textbf{Input:} safety checking problem \((\text{Init}, \text{Tr}, \text{Bad})\)

\textbf{Output:} set of all MIVCs of \((\text{Init}, \text{Tr}, \text{Bad})\)

\begin{enumerate}
\item \(\text{MIVCs} \leftarrow \emptyset\)
\item \(\text{map} \leftarrow \top\)
\end{enumerate}

MCS/MUA Extraction Loop

\begin{enumerate}
\item
\item
\item
\item
\item
\item
\item
\item
\item
\item
\item
\item
\end{enumerate}

Seed Extraction Loop
(all seeds guaranteed \texttt{SAFE})

\begin{enumerate}
\item \textbf{return} \(\text{MIVCs}\)
\end{enumerate}
We can find MCSes/MUAs using Unreachability Debugging [1][2]

- Enhanced TR $T_{ren}$: Add a mux at each gate output with select line $e_i$
- The select line is the output of a constant register that is assigned either 0 or 1 as part of the initial state assignment
- Enhanced initial states: $Init_{en} = Init \land AtMost(N, e_1, ..., e_n)$
- Solve $(Init_{en}, T_{ren}, Bad)$ for $N = 1, 2, 3, ...$
- Counter-example indicates an MCS of cardinality $N$

MARCO versus CAMIVC

• Anytime performance
  – MARCO finds MIVCs early and often throughout its run
  – CAMIVC must find all MCSes/MUAs first, which may be intractable

• Overall performance
  – MARCO must check each seed for safety with IC3 – very costly
  – CAMIVC does not need to check seeds
  – However, it does use IC3 to find MCSes – also very costly

• Both algorithms find all MCSes/MUAs and MIVCs
MARCO versus CAMIVC

• Anytime performance
  – MARCO finds MIVCs early and often throughout its run
  – CAMIVC must find all MCSes/MUAs first, which may be intractable

• Overall performance
  – MARCO must check each seed for safety with IC3 – very costly
  – CAMIVC does not need to check seeds
  – However, it does use IC3 to find MCSes – also very costly

• Both algorithms find all MCSes/MUAs and MIVCs

Iterations of MARCO seed extraction loop

= 

Iterations of CAMIVC phase 1 + Iterations of CAMIVC phase 2
The trade-off is just the result of shifting computation around

- MARCO finds seeds in an arbitrary order
- CAMIVC finds all UNSAFE seeds and then all SAFE ones

The UMIVC algorithm

- Truncated MCS/MUA extraction loop finds all MCSes of size $k$ or less (no safety checks)
- Fewer iterations of the seed extraction loop (with safety checks)

Subsumes MARCO and CAMIVC

- In MARCO, $k = 0$
- In CAMIVC, $k = \infty$
Algorithm 3 UMIVC

Input: safety checking problem \((\text{Init}, \text{Tr}, \text{Bad}), k \in \mathbb{Z} \cup \\{\infty\}\)
Output: set of all MIVCs of \((\text{Init}, \text{Tr}, \text{Bad})\)

1: \(\textit{MIVCs} \leftarrow \emptyset\)
2: \(\textit{map} \leftarrow \top\)
3: for \(i = 1\) to \(k\) do
4: \hspace{1em} while more MCSes of cardinality \(i\) exist do
5: \hspace{2em} \(\textit{mcs} \leftarrow \text{FindMCS}(\text{Init}, \text{Tr}, \text{Bad}, i)\)
6: \hspace{2em} \(\textit{map} \leftarrow \textit{map} \wedge \text{blockDown}(\text{Tr} \setminus \textit{mcs})\)
7: \hspace{1em} end while
8: end for
9: while \(\textit{map}\) is SAT do
10: \hspace{1em} \(\textit{seed} \leftarrow \text{getUnexplored}(\textit{map})\)
11: \hspace{1em} if \(k = \infty\) or \((\text{Init}, \text{seed}, \text{Bad})\) is SAFE then
12: \hspace{2em} \(\textit{mivc} \leftarrow \text{shrink}(\textit{seed})\)
13: \hspace{2em} \(\textit{MIVCs} \leftarrow \textit{MIVCs} \cup \{\textit{mivc}\}\)
14: \hspace{2em} \(\textit{map} \leftarrow \textit{map} \wedge \text{blockUp}(\textit{mivc})\)
15: \hspace{1em} else
16: \hspace{2em} \(\textit{mua} \leftarrow \text{grow}(\textit{seed})\)
17: \hspace{2em} \(\textit{map} \leftarrow \textit{map} \wedge \text{blockDown}(\textit{mua})\)
18: \hspace{1em} end if
19: end while
20: return \(\textit{MIVCs}\)
Algorithm 3 UMIVC

**Input:** safety checking problem \((Init, Tr, Bad), k \in \mathbb{Z} \cup \{\infty\}\)

**Output:** set of all MIVCs of \((Init, Tr, Bad)\)

1: \(MIVCs \leftarrow \emptyset\)
2: \(map \leftarrow \top\)

---

**Truncated MCS/MUA Extraction Loop**

---

**Seed Extraction Loop**

---

20: \textbf{return} \(MIVCs\)
Why UMIVC?

• Why do we need UMIVC?
  – Phase 1 iteration: find an MCS/MUA
  – Phase 2 iteration: find a seed, check for safety, grow or shrink
  – Certain optimizations allow phase 1 to avoid IC3 altogether – *significantly* more efficient than phase 2

• Does UMIVC also apply to MUS enumeration?
  – Technically yes
  – SAT checks on seeds are not nearly as expensive as IC3
  – Phase 2 is simply not that big of a problem in the MUS domain

Safety checking: cheap phase 1, expensive phase 2

SAT: moderate phase 1, moderate phase 2
Optimizations

- The grow procedure
  - Instead of brute force, find an MCS using unreachability debugging and find the complement
  - Runtime for grow is comparable to finding an MCS in the first phase

- Caching invariants and counter-examples in LRU caches
  - Check cached counter-examples, then invariants, then safety check
  - Checking invariants is still expensive, so we only store one

- Biased safety checking
  - Safety checking uses caches, BMC, and IC3
  - Maximal seeds are likely to be SAFE (especially early), minimal UNSAFE
  - Execute checks in a different order depending on expected outcome
Optimizations

• Approximate MCS extraction

• Truncated MCS enumeration is still expensive, since unreachability debugging uses IC3

• The algorithm is still correct even if this phase doesn’t find all MCSes or finds non-minimal correction subsets
  – Does not apply to CAMIVC, need to check for safety in phase 2

• Just skip IC3 and rely entirely on BMC in the debug algorithm
  – We still find correction subsets
  – Not necessarily minimal ones
  – Not necessarily all of them
# Variants

<table>
<thead>
<tr>
<th>Variant</th>
<th>k</th>
<th>Exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAMIVC</td>
<td>$\infty$</td>
<td>Bottom-up</td>
</tr>
<tr>
<td>MARCO</td>
<td>0</td>
<td>Arbitrary</td>
</tr>
<tr>
<td>MARCO-DOWN</td>
<td>0</td>
<td>Top-down</td>
</tr>
<tr>
<td>MARCO-UP</td>
<td>0</td>
<td>Bottom-up</td>
</tr>
<tr>
<td>MARCO-ZIGZAG</td>
<td>0</td>
<td>Zig-zagging</td>
</tr>
<tr>
<td>k-UMIVC</td>
<td>$k$</td>
<td>Arbitrary</td>
</tr>
<tr>
<td>k-UMIVC-DOWN</td>
<td>$k$</td>
<td>Top-down</td>
</tr>
<tr>
<td>k-UMIVC-UP</td>
<td>$k$</td>
<td>Bottom-up</td>
</tr>
<tr>
<td>k-UMIVC-ZIGZAG</td>
<td>$k$</td>
<td>Zig-zagging</td>
</tr>
</tbody>
</table>
Outline

• Motivation
• Background
• The UMIVC Algorithm
  • Experiments
• Conclusion
Experiments

• Everything is open source
  – https://github.com/ryanberryhill/pme
  – Documentation is sparse

• Two sets of experiments
  – Many configurations executed on 50 HWMCC 2011 circuits
    • MIVC enumeration is often tractable, more results
  – Best configurations executed on all 181 SAFE instances from HWMCC17
    • Often too difficult to find MIVCs

• All experiments have a 15 minute timeout period
Experiments – HWMCC11

• Intended to quickly evaluate a wide range of configurations

• All optimizations appear to improve performance

• First phase efficiency
  – In one configuration with \( k = 3 \), we observe the first phase finds 86x as many correction subsets as the second in 3.6x the time
    • Not exactly a fair comparison, the second phase finds MCSes
  – Similar for other configurations

<table>
<thead>
<tr>
<th></th>
<th>Correction Subsets</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1</td>
<td>32 048</td>
<td>11 630</td>
</tr>
<tr>
<td>Phase 2</td>
<td>372</td>
<td>3 193</td>
</tr>
</tbody>
</table>
### Experiments – HWMCC17

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Phase 1 complete</th>
<th>Found One MIVC</th>
<th>Found All MIVCs</th>
<th>Found SMIVC</th>
<th>Total MIVCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-UMIVC-UP</td>
<td>55</td>
<td>49</td>
<td>32</td>
<td>49</td>
<td>6907</td>
</tr>
<tr>
<td></td>
<td>12265</td>
<td>16869</td>
<td>30075</td>
<td>16869</td>
<td></td>
</tr>
<tr>
<td>2-UMIVC-DOWN</td>
<td>63</td>
<td>61</td>
<td>37</td>
<td>37</td>
<td>1866</td>
</tr>
<tr>
<td></td>
<td>3739</td>
<td>7100</td>
<td>26887</td>
<td>26874</td>
<td></td>
</tr>
<tr>
<td>3-UMIVC-ZZ</td>
<td>55</td>
<td>54</td>
<td>33</td>
<td>46</td>
<td>2434</td>
</tr>
<tr>
<td></td>
<td>12260</td>
<td>15298</td>
<td>30294</td>
<td>20457</td>
<td></td>
</tr>
<tr>
<td>CAMUS</td>
<td>32</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>30819</td>
<td>31462</td>
<td>31472</td>
<td>31462</td>
<td></td>
</tr>
</tbody>
</table>

- 64 Circuits for which at least one MIVC was found
- Note: model checking takes only 481 s for these 64 instances
- MIVC enumeration is significantly harder
Experiments – HWMCC17

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Phase 1 complete</th>
<th>Found One MIVC</th>
<th>Found All MIVCs</th>
<th>Found SMIVC</th>
<th>Total MIVCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-UMIVC-UP</td>
<td>55</td>
<td>49</td>
<td>32</td>
<td>49</td>
<td>6907</td>
</tr>
<tr>
<td></td>
<td>12265</td>
<td>16869</td>
<td>30075</td>
<td>16869</td>
<td></td>
</tr>
<tr>
<td>2-UMIVC-DOWN</td>
<td>63</td>
<td>61</td>
<td>37</td>
<td>37</td>
<td>1866</td>
</tr>
<tr>
<td></td>
<td>3739</td>
<td>7100</td>
<td>26887</td>
<td>26874</td>
<td></td>
</tr>
<tr>
<td>3-UMIVC-ZZ</td>
<td>55</td>
<td>54</td>
<td>33</td>
<td>46</td>
<td>2434</td>
</tr>
<tr>
<td></td>
<td>12260</td>
<td>15298</td>
<td>30294</td>
<td>30457</td>
<td></td>
</tr>
<tr>
<td>CAMUS</td>
<td>32</td>
<td>30</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30819</td>
<td>31462</td>
<td>31472</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 64 Circuits for which at least one MIVC was found
- Note: model checking takes only 481 s for these 64 instances
- MIVC enumeration is significantly harder

Best anytime performance
Experiments – HWMCC17

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Phase 1 complete</th>
<th>Found One MIVC</th>
<th>Found All MIVCs</th>
<th>Found SMIVC</th>
<th>Total MIVCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-UMIVC-UP</td>
<td>55</td>
<td>49</td>
<td>32</td>
<td>49</td>
<td>6907</td>
</tr>
<tr>
<td></td>
<td>12265</td>
<td>16869</td>
<td>30075</td>
<td>16869</td>
<td></td>
</tr>
<tr>
<td>2-UMIVC-DOWN</td>
<td>63</td>
<td>61</td>
<td>37</td>
<td>37</td>
<td>1866</td>
</tr>
<tr>
<td></td>
<td>3739</td>
<td>7100</td>
<td>26887</td>
<td>26874</td>
<td></td>
</tr>
<tr>
<td>3-UMIVC-ZZ</td>
<td>55</td>
<td>54</td>
<td>33</td>
<td>46</td>
<td>2434</td>
</tr>
<tr>
<td></td>
<td>12260</td>
<td>15298</td>
<td>30294</td>
<td>20457</td>
<td></td>
</tr>
<tr>
<td>CAMUS</td>
<td>32</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>30819</td>
<td>31462</td>
<td>31472</td>
<td>31462</td>
<td></td>
</tr>
</tbody>
</table>

- 64 Circuits for which at least one MIVC was found
- Note: model checking takes only 481 s for these 64 instances
- MIVC enumeration is significantly harder
### Experiments – HWMCC17

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Phase 1 complete</th>
<th>Found One MIVC</th>
<th>Found All MIVCs</th>
<th>Found SMIVC</th>
<th>Total MIVCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-UMIVC-UP</td>
<td>55</td>
<td>49</td>
<td>32</td>
<td>49</td>
<td>6907</td>
</tr>
<tr>
<td></td>
<td>12265</td>
<td>16869</td>
<td>30075</td>
<td>16869</td>
<td></td>
</tr>
<tr>
<td>2-UMIVC-DOWN</td>
<td>63</td>
<td>61</td>
<td>37</td>
<td>37</td>
<td>1866</td>
</tr>
<tr>
<td></td>
<td>3739</td>
<td>7100</td>
<td>26887</td>
<td>26874</td>
<td></td>
</tr>
<tr>
<td>3-UMIVC-ZZ</td>
<td>55</td>
<td>54</td>
<td>33</td>
<td>46</td>
<td>2434</td>
</tr>
<tr>
<td></td>
<td>12260</td>
<td>15298</td>
<td>30294</td>
<td>20457</td>
<td></td>
</tr>
<tr>
<td>CAMUS</td>
<td></td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>107</td>
</tr>
</tbody>
</table>

- 64 Circuits for which at least one MIVC was found
- Note: model checking takes only 481 s for these 64 instances
- MIVC enumeration is significantly harder
### Experiments – HWMCC17

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Phase 1 complete</th>
<th>Found One MIVC</th>
<th>Found All MIVCs</th>
<th>Found SMIVC</th>
<th>Total MIVCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-UMIVC-UP</td>
<td>55</td>
<td>49</td>
<td>32</td>
<td>49</td>
<td>6907</td>
</tr>
<tr>
<td></td>
<td>12265</td>
<td>16869</td>
<td>30075</td>
<td>16869</td>
<td></td>
</tr>
<tr>
<td>2-UMIVC-DOWN</td>
<td>63</td>
<td>61</td>
<td>37</td>
<td>37</td>
<td>1866</td>
</tr>
<tr>
<td></td>
<td>3739</td>
<td>7100</td>
<td>26887</td>
<td>26874</td>
<td></td>
</tr>
<tr>
<td>3-UMIVC-ZZ</td>
<td>55</td>
<td>54</td>
<td>33</td>
<td>46</td>
<td>2434</td>
</tr>
<tr>
<td></td>
<td>12260</td>
<td>15298</td>
<td>30294</td>
<td>20457</td>
<td></td>
</tr>
<tr>
<td>CAMUS</td>
<td>32</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>30819</td>
<td>31462</td>
<td>31472</td>
<td>31462</td>
<td></td>
</tr>
</tbody>
</table>

- 64 Circuits for which at least one MIVC was found
- Note: model checking takes only 481 s for these 64 instances
- MIVC enumeration is significantly harder
Experiments – HWMCC17

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Phase 1 complete</th>
<th>Found One MIVC</th>
<th>Found All MIVCs</th>
<th>Found SMIVC</th>
<th>Total MIVCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-UMIVC-UP</td>
<td>55</td>
<td>49</td>
<td>32</td>
<td>49</td>
<td>6907</td>
</tr>
<tr>
<td></td>
<td>12265</td>
<td>16869</td>
<td>30075</td>
<td>16869</td>
<td></td>
</tr>
<tr>
<td>2-UMIVC-DOWN</td>
<td>63</td>
<td>61</td>
<td>37</td>
<td>37</td>
<td>1866</td>
</tr>
<tr>
<td></td>
<td>3739</td>
<td>7100</td>
<td>26887</td>
<td>26874</td>
<td></td>
</tr>
<tr>
<td>3-UMIVC-ZZ</td>
<td>55</td>
<td>54</td>
<td>33</td>
<td>46</td>
<td>2434</td>
</tr>
<tr>
<td></td>
<td>12260</td>
<td>15298</td>
<td>30294</td>
<td>20457</td>
<td></td>
</tr>
<tr>
<td>CAMUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>107</td>
</tr>
</tbody>
</table>

- 64 Circuits for which at least one MIVC was found
- Note: model checking takes only 481 s for these 64 instances
- MIVC enumeration is significantly harder
### Experiments – HWMCC17

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Phase 1 complete</th>
<th>Found One MIVC</th>
<th>Found All MIVCs</th>
<th>Found SMIVC</th>
<th>Total MIVCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-UMIVC-UP</td>
<td>55</td>
<td>49</td>
<td>32</td>
<td>49</td>
<td>6907</td>
</tr>
<tr>
<td></td>
<td>12265</td>
<td>16869</td>
<td>30075</td>
<td>16869</td>
<td></td>
</tr>
<tr>
<td>2-UMIVC-DOWN</td>
<td>63</td>
<td>61</td>
<td>37</td>
<td>37</td>
<td>1866</td>
</tr>
<tr>
<td></td>
<td>3739</td>
<td>7100</td>
<td>26887</td>
<td>26874</td>
<td></td>
</tr>
<tr>
<td>3-UMIVC-ZZ</td>
<td>55</td>
<td>54</td>
<td>33</td>
<td>46</td>
<td>2434</td>
</tr>
<tr>
<td></td>
<td>12260</td>
<td>15298</td>
<td>30294</td>
<td>20457</td>
<td></td>
</tr>
<tr>
<td>CAMUS</td>
<td>32</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>30819</td>
<td>31462</td>
<td>31472</td>
<td>31462</td>
<td></td>
</tr>
</tbody>
</table>

- 64 Circuits for which at least one MIVC was found
- Note: model checking takes only 481 s for these 64 instances
- MIVC enumeration is significantly harder
Outline

• Motivation
• Background
• The UMIVC Algorithm
• Experiments
• Conclusion
Conclusion

• MIVCs are more human-understandable than other certificates of safety

• Closely-related to finding MUSes
  – MARCO and CAMUS apply and are two extremes of a trade-off curve

• UMIVC subsumes MARCO and CAMIVC
  – Better control over the anytime/overall trade-off
  – Optimizations that make UMIVC perform better than either algorithm

• Results for HWMCC11 and HWMCC17 circuits
  – Of 181 SAFE HWMCC17 circuits, an MIVC is found for 64