$k$-FAIR = $k$-LIVENESS + FAIR
Revisiting SAT-based Liveness Algorithms

FMCAD’18

Alexander Ivrii
Ziv Nevo
Jason Baumgartner
IBM
Personal perspective

• Support for *liveness properties* in IBM’s formal hardware verification tool:
  • BDD-based algorithms (Clarke-Grumberg-Peled, ’99)
    • Compute all reachable states
    • Less scalable but occasionally useful
  • Liveness-to-Safety translation (Biere-Artho-Schuppan, FMICS’02)
    • Replaces verification of liveness properties by verification of safety properties
    • Extremely useful
  • FAIR (Bradley-Somenzi-Hassan-Zhang, FMCAD’11)
    • Will describe in a minute
    • Seems extremely interesting but also hard to implement
  • *k*-LIVENESS (Claessen-Sörensson, FMCAD’12)
    • Will describe in a minute
    • Quite useful
  • *k*-FAIR = *k*-LIVENESS + FAIR
    • Will describe in a minute
    • Quite useful
Liveness properties

• Consider liveness properties of the form $FGq$
  • “on every trace $q$ eventually becomes true forever”

• A counterexample to $FGq$ is an infinite trace on which $q$ becomes false infinitely often
  • Commonly represented as a lasso-shaped trace:

```
Initial states

¬q-state s

prefix

repeating loop suffix
```
Reachability and stabilizing constraints

• A reachability constraint $R$ indicates that all states on a potential lasso-shaped counterexample belong to $R$.

• A stabilizing constraint $S$ indicates that all states on the loop of a potential lasso-shaped counterexample belong to $S$. 
**k-LIVENESS** (Claessen-Sörensson, FMCAD’12)

- *k*-LIVENESS tries to bound the number of times that \( q \) can become false on any trace.

- Start with \( k = 1 \).
- **Make safety query**: is there a trace with at least \( k \) occurrences of \( \neg q \)?

- If such a trace does not exist: \( \mathsf{FG}q \) passes.
- If such a trace exists **and** has a repeated \( \neg q \)-state: \( \mathsf{FG}q \) fails.
- Otherwise: increase \( k \).

- Any safety model checker can be used (capable of proving properties).
- Using IC3 offers advantage due to close relation between queries for different \( k \).
FAIR (Bradley-Somenzi-Hassan-Zhang, FMCAD’11)

- The presented variant is specialized to properties of the form $\text{FG}_q$, and is slightly less general (but simpler).

- FAIR incrementally learns reachability and stabilizing constraints on the state space.

- **Make SAT query:** “is there a $\neg q$-state $s$, subject to previously discovered reachability and stabilizing constraints?”

- If such a state does not exist, $\text{FG}_q$ passes.
Let’s suppose such a state $s$ exists. Let’s check if $s$ can be completed to a lasso-shaped counterexample.

- **Make safety query:** “is $s$ reachable from initial states, subject to previously discovered reachability constraints?”
- **Make safety query:** “does $s$ have a loop to itself, subject to previously discovered reachability and stabilizing constraints?”

---

FAIR (Bradley-Somenzi-Hassan-Zhang, FMCAD’11)

continued…
FAIR (Bradley-Somenzi-Hassan-Zhang, FMCAD’11)

- If both safety queries are satisfiable: $FGq$ fails.

- For unsatisfiable safety queries, the model checker needs to be capable of producing inductive proofs of unsatisfiability (commonly IC3 is used).

- If $s$ is not reachable from initial states:
  - The inductive proof represents a new reachability constraint (excluding $s$).
  - If $s$ does not have a loop to itself:
    - We can generalize $s$ to a larger set of states without a self-loop.
    - The complement of this set represents a new stabilizing constraint (excluding $s$).

- In either case, the algorithm makes progress.
$k$-Liveness vs. FAIR

• The two algorithms have different strengths:
  
  • $k$-LIVENESS works well when a small value of $k$ is sufficient (otherwise, underlying safety queries usually become complicated).
  
  • FAIR works well when inductive proofs restrict large portions of state space (otherwise, many iterations are required).

• The presented algorithm $k$-FAIR combines ideas from both approaches.
$k$-FAIR (the feature presentation)

- Start with $k = 1$.
- **Make safety query:** “is there a trace with at least $k$ occurrences of $\neg q$, subject to previously discovered reachability and stabilizing constraints?”

If such a trace does not exist: **$FGq$ passes.**

If such a trace exists **and** has a repeated $\neg q$-state: **$FGq$ fails.**

Initial states

$\neg q$-state

$\neg q$-state

$\neg q$-state

stabilizing constraints should hold here

reachability constraints should hold here

continued...
**k-FAIR (the feature presentation)**

**One option:**
- Increase $k$.

**Another option:**
- Select any $\neg q$-state $s$ from the trace.
- **Make safety query:** “does $s$ have a loop to itself, subject to previously discovered reachability and stabilizing constraints?”
  - If $s$ has a loop to itself: $\text{FGq fails}$.
  - If $s$ does not have a loop to itself: extract a new stabilizing constraint (excluding $s$).
  - We have a choice to increase $k$, leave $k$ the same, or even to decrease $k$. 
Finding additional stabilizing constraints

• Both $k$-LIVENESS and FAIR have an important optimization of looking for single-literal stabilizing constraints.

• There are certain differences:
  • The technique in $k$-LIVENESS is stronger, but is used only as preprocessing.
    • Considers all nets in the circuit as candidates; uses liveness signal in a stronger way.
  • FAIR looks for new constraints periodically.
    • Leverages new reachability and stabilizing constraints.

• $k$-FAIR uses the best of both worlds:
  • Uses the stronger technique; can apply this technique as preprocessing and periodically.
  • Minor technicality: how to pass stabilizing constraints to the safety model checker?
    • See the paper for details
Experiments: setup & configurations

- All single-property liveness benchmarks from HWMCC’11 – HWMCC’17, and various in-house
- Preprocessed using standard logic synthesis techniques (à la ABC)
- 3 hours time-limit

- $k$-FAIR Configurations:

<table>
<thead>
<tr>
<th></th>
<th>Stabilizing Constraints</th>
<th>Increase $k$</th>
<th>Look for self-loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAIR</td>
<td>Periodically</td>
<td>Never</td>
<td>Every iteration</td>
</tr>
<tr>
<td>Combined-50</td>
<td>Periodically</td>
<td>Every 50th iteration</td>
<td>Every iteration</td>
</tr>
<tr>
<td>Combined-5</td>
<td>Periodically</td>
<td>Every 5th iteration</td>
<td>Every iteration</td>
</tr>
<tr>
<td>Improved $k$-LIVENESS</td>
<td>Periodically</td>
<td>Every iteration</td>
<td>Never</td>
</tr>
<tr>
<td>Standard $k$-LIVENESS</td>
<td>Only initially</td>
<td>Every iteration</td>
<td>Never</td>
</tr>
</tbody>
</table>

- LTS-BMC: Liveness-to-Safety, followed by Bounded Model Checking
- LTS-IC3: Liveness-to-Safety, followed by IC3
Experiments: results

<table>
<thead>
<tr>
<th></th>
<th>PASS solved</th>
<th>PASS time</th>
<th>FAIL solved</th>
<th>FAIL time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAIR</td>
<td>108</td>
<td>338,475</td>
<td>89</td>
<td>351,634</td>
</tr>
<tr>
<td>Combined-50</td>
<td>111</td>
<td>301,592</td>
<td>94</td>
<td>321,260</td>
</tr>
<tr>
<td>Combined-5</td>
<td>122</td>
<td>166,655</td>
<td>104</td>
<td>240,458</td>
</tr>
<tr>
<td>Improved $k$-LIVENESS</td>
<td>123</td>
<td>173,077</td>
<td>97</td>
<td>245,543</td>
</tr>
<tr>
<td>Standard $k$-LIVENESS</td>
<td>117</td>
<td>250,431</td>
<td>100</td>
<td>225,971</td>
</tr>
<tr>
<td>LTS-BMC</td>
<td>–</td>
<td>–</td>
<td>114</td>
<td>94,103</td>
</tr>
<tr>
<td>LTS-IC3</td>
<td>116</td>
<td>225,059</td>
<td>99</td>
<td>226,321</td>
</tr>
<tr>
<td>Virtual Best</td>
<td>131</td>
<td>37,270</td>
<td>117</td>
<td>29,315</td>
</tr>
</tbody>
</table>

- best configuration
- best additional value
Experiments: conclusions

• Falsification:
  • Liveness-to-Safety followed by BMC is a very strong strategy
  • Combined-5 brings most additional value
  • Running all 7 configurations significantly improves performance further

• Proof:
  • Improved $k$-LIVENESS performs best overall
  • Liveness-to-safety followed by IC3 brings most additional value
  • Running other configurations did not further improve performance at all

• **What we want to be true, but experiments don’t show:**
  • Intuitively, $k$-FAIR should especially help when $\text{FGq}$ holds:
    • “When $k$-LIVENESS gets stuck, we can stop increasing $k$ and apply FAIR”.
  • In retrospect, the considered $k$-FAIR configurations are not very intelligent.
Comparison of FAIR in $k$-FAIR and in IIMC

<table>
<thead>
<tr>
<th></th>
<th>PASS solved</th>
<th>PASS unique solved</th>
<th>PASS time</th>
<th>FAIL solved</th>
<th>FAIL unique solved</th>
<th>FAIL time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAIR in $k$-FAIR</td>
<td>108</td>
<td>18</td>
<td>208,875</td>
<td>89</td>
<td>21</td>
<td>70,834</td>
</tr>
<tr>
<td>FAIR in IIMC</td>
<td>101</td>
<td>11</td>
<td>277,657</td>
<td>70</td>
<td>2</td>
<td>242,762</td>
</tr>
</tbody>
</table>

- Both variants have unique value
- Overall, FAIR in $k$-FAIR performs substantially better than FAIR in IIMC
  - However, the improvements may be due to a large number of different factors
  - Detailed comparison is difficult
- The (simplified) variant of FAIR presented here is a viable alternative to the implementation in IIMC
Summary:

• **SAT-based liveness is a pure FMCAD-driven research.**

• **Quick takeaway:** periodically detecting single-literals stabilizing constraints improves $k$-LIVENESS, and is simple to implement.

• **A slightly less quick takeaway:** combining the strengths of $k$-LIVENESS and FAIR brings unique value, and is not too difficult to implement.

• **There is a lot of room for further improvement:** devising better $k$-FAIR strategies, carefully balancing the efforts spent in different $k$-FAIR components, improving underlying IC3 implementation towards safety queries posed by $k$-FAIR, etc.

Thank You!