Heuristic-Guided Counterexample Search in FLAVERS

Jianbin Tan, George Avrunin, Lori Clarke, Shlomo Zilberstein
University of Massachusetts, Amherst

Stefan Leue
University of Konstanz
Finite State Verification (FSV)

- FSV techniques verify whether a model of a system is consistent with a specified property.
  - If the property is found to be violated, *counterexamples* are usually provided to demonstrate how the violation happened.
  - Counterexamples help isolate the cause of the problem.
Counterexample Search

- Can represent the verification problem as a search for counterexamples
  - Two metrics: time and length
- Standard algorithms have drawbacks
  - BFS: finds the shortest counterexample but usually is slow
  - DFS: usually is fast, but tends to produce a long counterexample
- Want a heuristic search algorithm that usually finds short counterexamples fast
Outline

- FLAVERS overview
- Heuristic search algorithms considered
- Experimental results
- Related work
- Conclusions and future work
FLAVERS Architecture

Flow Analysis for Verification of Systems

- Property
- System
- Model
- Constraints

Verification algorithm

Report conclusive result (property holds)

Report inconclusive result (property might not hold) and return a counterexample
Property

- Specifies sequences of events that should occur on all executions of the system
- Represented as a finite-state automaton (FSA)
- Example: “lock” can never occur consecutively
Model

- A flow graph that models the event sequences of the system
  - Built from annotated control flow graphs for the threads
  - Each node may be labeled by one event
  - Each path in the model represents a sequence of events
  - Conservative but imprecise
Model: An Example

Task1
loop
if ( locked ) then
  call Task2.unlock
else
  call Task2.lock
end if
exit when done
end loop

Task2
loop
select
accept lock
  locked:=true
or
accept unlock
  locked:=false
end select
exit when done
end loop
Model: An Example

Task1
loop
if ( locked ) then
call Task2.unlock
else
   call Task2.lock
end if
exit when done
end loop

Task2
loop
select
   accept lock
   locked:=true
or
   accept unlock
   locked:=false
end select
exit when done
end loop
Model: An Example

Task1
loop
if ( locked ) then
    call Task2.unlock
else
    call Task2.lock
end if
exit when done
end loop

Task2
loop
select
    accept lock
    locked:=true
or
    accept unlock
    locked:=false
end select
exit when done
end loop
Model: An Example

Task1
loop
if ( locked ) then
call Task2.unlock
else
call Task2.lock
end if
exit when done
end loop

Task2
loop
select
accept lock
locked:=true
or
accept unlock
locked:=false
end select
exit when done
end loop
Model is Imprecise

Task1 loop
if ( locked ) then
call Task2.unlock
else
call Task2.lock
end if
exit when done
end loop
unlock

Task2 loop
select
accept lock
locked:=true
or
accept unlock
locked:=false
end select
exit when done
end loop

L==t
L==f
lock
unlock
L:=t
L:=f
Constraints

- Introduced to refine the model
  - Specify valid sequences of events in the model
  - If a path is not accepted by a constraint, the path is rejected
- Represented as FSAs
- Several kinds of constraints
  - Many can be automatically created
Constraint: An Example

Control flow graph of Task1

Task Automaton (TA) of Task1
Constraints Make the Model More Precise
Verification Algorithms

- FLAVERS explores all paths in the model that do not violate any constraint
- There are several alternative algorithms that can be used
  - Data-flow analysis algorithms work well when the property turns out to hold
  - Search algorithms work well when there are counterexamples
Search Framework

- Builds and checks a *node-tuple graph* on-the-fly

\[
( x, \langle p_1, c_2, \ldots, c_m \rangle )
\]

A node from the flow graph

A vector with states from the property FSA and each constraint FSA
(0, <1, t0>)
(1, <1, t1>)
(4, <1, t4>)
A violating node-tuple
The Search Framework

Put the initial node-tuple in the worklist \( W \)
While \( W \) is not Empty
  remove a node-tuple \( n \) from \( W \)
  for each successor \( s \) of \( n \)
    If \( s \) is a violating node-tuple
      Generate the counterexample
      Return INCONCLUSIVE
    Else if \( s \) has not been visited before
      Add \( s \) to \( W \)
  Return CONCLUSIVE
The Search Framework

Put the initial node-tuple in the worklist \( W \)
While \( W \) is not Empty
  remove a node-tuple \( n \) from \( W \)
  for each successor \( s \) of \( n \)

Consider different ways to remove elements from the worklist
- BFS: FILO
- DFS: FIFO
- Heuristic search: remove the node-tuple with the smallest value of an evaluation function \( f(n) \)
Considered Two Ways to Construct Evaluation Function $f$

- **Best First (BF):** $f(n) = h(n)$
- **Weighted A* (WA*):** $f(n) = g(n) + w^* h(n)$

Where:

- $h(n)$: a heuristic function that estimates distance from current node $n$ to a goal node
- $g(n)$: a function that gives a distance from the initial node to the current node
- $w$: a parameter that provides control over the trade-off between search time and the length of the path
Explore Heuristic Functions

- Usually based on aspects of the goal node
  - In FLAVERS, a goal node is a violating node-tuple
- Evaluated two heuristic functions that estimate distance to a goal node
  - TA heuristic: based on the TA states in a node-tuple
  - Trap heuristic: based on the property state in a node-tuple
The TA Heuristic

- In a violating node-tuple, each TA must be in its final state
- Estimate the distance to a violating node-tuple
  - Sum over all TAs of the shortest distance $d$ from the current state to the final state
  - E.g.: $d(t1) = 4$, $d(t5) = 2$
A trap state is a non-accepting sink state
- Multiple trap states can be merged
- Once the property is in a trap state, it can never get into an accepting state
- Fact: all safety properties can be represented by an FSA with a trap state
- **Trap node-tuple**: a node-tuple with the property in the trap state
2-Stage Search Strategy

- 1st stage: from the initial node-tuple, try to find a short path to a trap node-tuple fast

A trap node-tuple
2-Stage Search Strategy

- **1\textsuperscript{st} stage**: from the initial node-tuple, try to find a **short** path to a trap node-tuple **fast**

- **2\textsuperscript{nd} stage**: from the trap node-tuple, try to find a path to a final node-tuple **fast**
2-Stage Search Strategy

- Path found in the 1st stage is used to understand the cause of the violation
- Path found in the 2nd stage is needed to be sure the whole path is a counterexample
Trap Heuristic for the 1st Stage

- Estimate the distance to a trap node-tuple
  - Use the shortest distance $d$ from the current property state to the trap state
  - E.g.: $d(1)=2$; $d(2)=1$; $d(3)=0$
Search Algorithms Evaluated

<table>
<thead>
<tr>
<th></th>
<th>BFS</th>
<th>DFS</th>
<th>$\text{WA}_{\text{ta}}$</th>
<th>$\text{BF}_{\text{ta}}$</th>
<th>$\text{BF}_{\text{trap}}$</th>
<th>$\text{BF}<em>{\text{trap}} + \text{WA}</em>{\text{ta}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Stage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>2-Stage 1st Stage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>2-Stage 2nd Stage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

- Trap heuristic, which is based on the property, can not be used in the WA* algorithm, which is based on the node-tuple graph.
### Search Algorithms Evaluated

|       | BFS | DFS | $WA_{ta}$  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Stage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$BF_{ta}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$BF_{trap}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$BF_{trap} + WA_{ta}$</td>
</tr>
<tr>
<td>2-Stage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2nd Stage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- X: $BF_{trap}$ is based on the property trap state, not the final node.
### Search Algorithms Evaluated

<table>
<thead>
<tr>
<th>2-Stages</th>
<th>BFS</th>
<th>DFS</th>
<th>$W_{A_{ta}}$ $w=1, 2, 3, 5, 9$</th>
<th>$BF_{ta}$</th>
<th>$BF_{trap}$</th>
<th>$BF_{trap} + W_{A_{ta}}$ $w=1, 2, 3, 5, 9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Stage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>2-Stage</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; Stage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Stage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

- X: DFS and $BF_{ta}$ tend to produce a long path
## Search Algorithms Evaluated

<table>
<thead>
<tr>
<th></th>
<th>BFS</th>
<th>DFS</th>
<th>$WA_{ta}$</th>
<th>$BF_{ta}$</th>
<th>$BF_{trap}$</th>
<th>$BF_{trap} + WA_{ta}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1-Stage</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>2-Stage</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1st Stage</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>2nd Stage</strong></td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- X: BFS and $WA_{ta}$ tend to be slow
Metrics

- Runtime ratio:
  \[
  \frac{\text{Runtime}}{\text{BFS runtime}}
  \]

- Prefix length ratio:
  - Prefix length: length from the initial node-tuple to the first trap node-tuple
    \[
    \frac{\text{Prefix length}}{\text{BFS prefix length}}
    \]
Subjects in the Experiment

- Widely studied concurrent systems

- Properties originally hold in the systems
  - For each property, find a minimal set of constraints that are necessary to prove the property
  - Remove one constraint from the minimal set to generate a subject for the experiment
    - $N$ subjects will be generated if the set has $N$ constraints
Subjects in the Experiment

- Remove small subjects that do not differentiate the performance of algorithms
- Remove large subjects if not all algorithms can handle them
Runtime Ratios of 1-Stage Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>w=1</th>
<th>w=2</th>
<th>w=3</th>
<th>w=5</th>
<th>w=9</th>
<th>BF_{ta}</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>0.139</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WA_{ta}</td>
<td>0.803</td>
<td>0.395</td>
<td>0.433</td>
<td>0.488</td>
<td>0.523</td>
<td>0.758</td>
</tr>
</tbody>
</table>
Prefix Length Ratios of 1-Stage Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>w=1</th>
<th>w=2</th>
<th>w=3</th>
<th>w=5</th>
<th>w=9</th>
<th>BFta</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>22.791</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WA_{ta}</td>
<td>0.803</td>
<td>0.986</td>
<td>1.020</td>
<td>1.054</td>
<td>1.079</td>
<td>1.097</td>
</tr>
</tbody>
</table>
Runtime Ratios of 2-Stage Algorithms

BFS
w=1
w=2
w=3
w=5
w=9

DFS

WA_\text{ta}
Runtime Ratios of 2-Stage Algorithms

BFS
w=1
w=2
w=3
w=5
w=9
w=1
w=2
w=3
w=5
w=9
WAta

DFS

BF\text{ta}
Runtime Ratios of 2-Stage Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>w=1</th>
<th>w=2</th>
<th>w=3</th>
<th>w=5</th>
<th>w=9</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WA_{ta}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF_{trap}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAta</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF_{trap}+WAta</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- BFtrap
- WAta
- BFS
- DFS

Graph shows runtime ratios for different algorithms and weight values.
Runtime Ratios of 2-Stage Algorithms

- BFS
- WA
- BF
- DFS

Comparison of runtime ratios for different algorithms and parameters.
Runtime Ratios of 2-Stage Algorithms

- BFS
- WA_{ta}
- WAta
- BF_{trap}
- WAta
- DFS
- BF_{ta}
- BF_{trap} + WAta
- BF_{trap} + WAta
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>w=1</th>
<th>w=2</th>
<th>w=3</th>
<th>w=5</th>
<th>w=9</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>0.760</td>
<td>0.772</td>
<td>0.954</td>
<td>0.985</td>
<td>1.008</td>
</tr>
<tr>
<td>BF&lt;sub&gt;trap&lt;/sub&gt;</td>
<td>0.811</td>
<td>0.809</td>
<td>0.847</td>
<td>0.883</td>
<td>0.887</td>
</tr>
<tr>
<td>BF&lt;sub&gt;trap&lt;/sub&gt;+WA&lt;sub&gt;ta&lt;/sub&gt;</td>
<td>0.809</td>
<td>0.847</td>
<td>0.883</td>
<td>0.887</td>
<td>0.893</td>
</tr>
<tr>
<td>WA&lt;sub&gt;ta&lt;/sub&gt;</td>
<td>0.231</td>
<td>0.220</td>
<td>0.760</td>
<td>0.772</td>
<td>0.954</td>
</tr>
<tr>
<td>BF&lt;sub&gt;trap&lt;/sub&gt;+WA&lt;sub&gt;ta&lt;/sub&gt;</td>
<td>0.231</td>
<td>0.220</td>
<td>0.760</td>
<td>0.772</td>
<td>0.954</td>
</tr>
</tbody>
</table>

Prefix Length Ratios of 2-Stage Algorithms
Runtime Ratios Comparison

1st stage: $B F_{\text{trap+}} \ W A_{\text{ta}} (w=1)$
2nd stage: $B F_{\text{ta}}$

$W A_{\text{ta}} (w=2)$
Prefix Length Ratios Comparison

1st stage: $\text{BF}_{\text{trap}} + \text{WA}_{\text{ta}} (w=1)$
2nd stage: $\text{BF}_{\text{ta}}$

$\text{WA}_{\text{ta}}(w=2)$
Summary

- The 2-stage algorithm with $\text{BF}_{\text{trap}} + \text{WA}_{\text{ta}} (w=1)$ and $\text{BF}_{\text{ta}}$ is surprisingly good
  - Runtime ratio:
    - Range from 0.001 to 0.903
    - Average 0.083
    - On average, faster than DFS (0.139)
  - Prefix length ratio:
    - Range from 0.021 to 1.278
    - Average 0.809
  - Works consistently well for these systems
Threats to Validity

- Systems used in the experiment might not be representative
- The inconclusive subjects are created by removing a constraint from the originally conclusive subjects
- Did not evaluate the performance of these algorithms in cases where the property FSAs do not have a trap state
  - 2-stage algorithm is not applicable in these cases
Related Work

- TA heuristic was first described by Cobleigh, etc.
  - Focused on comparing different algorithms used in different situations

- Our work developed the trap heuristic and the 2-stage search algorithm and focused on counterexamples
Related Work

- Apply heuristic search to guide the counterexample search in other FSV tools
  - HSF-SPIN: heuristics based on the property and the structure of the model
  - Java PathFinder: heuristics based on the structure of the model
  - MurØ: Hamming Distance based heuristic
  - VeriSoft: genetic algorithm

- Multi-stage search used in AI
Future Work

- Use heuristic algorithms on a broader range of systems and properties
  - Apply them to Java programs

- Explore the use of heuristic search to find counterexamples that are useful to refine the model
Conclusions

- Explored heuristic search algorithms to find short counterexample fast
- The best algorithm used property and model information
  - Always finds short, but not necessarily shortest, prefix faster than BFS and on average faster than DFS
- Other FSV approaches could also consider property and model based 2-stage heuristic search algorithms
Thank You

Questions?
Observation

- **Trap node-tuple**: a node-tuple with the property in the trap state
  - Use the trap state to guide the search to a trap node-tuple ("first part")
  - Once at a trap node-tuple, start a new search for a violating node-tuple that examines the successors of the trap node-tuple only ("second part")
  - Need second part to be sure it is a counterexample, but usually only need first part to understand the cause
Use the number of transitions to the trap state to reduce the tie:
- For a property state that has \( k>1 \) transitions to the trap state: \( d = 1 + 1/k \)
- More transitions mean more possibilities to enter the trap state
- Small estimated value is preferred

\[
\begin{align*}
d(0) &= 2 \\
d(1) &= 1.5 \\
d(2) &= h(3) = 1.333 \\
d(4) &= 0
\end{align*}
\]
Runtime Ratios of 2-Stage Algorithms (2\textsuperscript{nd} Stage uses DFS)

<table>
<thead>
<tr>
<th></th>
<th>w=1</th>
<th>w=2</th>
<th>w=3</th>
<th>w=5</th>
<th>w=9</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>0.291</td>
<td>0.277</td>
<td>0.295</td>
<td>0.347</td>
<td>0.389</td>
</tr>
<tr>
<td>(W_{A_{ta}})</td>
<td>0.131</td>
<td>0.125</td>
<td>0.129</td>
<td>0.137</td>
<td>0.163</td>
</tr>
</tbody>
</table>
Runtime Ratios of 2-Stage Algorithms (2\textsuperscript{nd} Stage uses BF\textsubscript{ta})

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$w=1$</th>
<th>$w=2$</th>
<th>$w=3$</th>
<th>$w=5$</th>
<th>$w=9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>0.231</td>
<td>0.220</td>
<td>0.240</td>
<td>0.281</td>
<td>0.324</td>
</tr>
<tr>
<td>WA\textsubscript{ta}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF\textsubscript{trap}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.092</td>
</tr>
<tr>
<td>BF\textsubscript{trap}+WA\textsubscript{ta}</td>
<td></td>
<td></td>
<td></td>
<td>0.084</td>
<td>0.091</td>
</tr>
</tbody>
</table>

Notes:
- Runtime ratios are shown on a logarithmic scale.
- The graph compares the performance of different algorithms.
- The algorithms are BFS, WA\textsubscript{ta}, BF\textsubscript{trap}, and BF\textsubscript{trap}+WA\textsubscript{ta}.
- The $w$ values represent different parameters affecting the algorithms.