Searching for Liveness Property Violations in Concurrent Systems with ACO

Francisco Chicano and Enrique Alba
• Concurrent software is difficult to test ...

• ... and it is in the heart of a lot of critical systems

• Techniques for proving the correctness of concurrent software are required

• Model checking → fully automatic

• In the past the work using metaheuristics focused on safety properties

• In this work we focus on liveness properties
**Explicit State Model Checking**

- **Objective**: Prove that model $M$ satisfies the property $\mathcal{F}$: $M \models \mathcal{F}$
- **HSF-SPIN**: the property $\mathcal{F}$ is an LTL formula

**Model $M$**

**LTL formula** $\neg \mathcal{F}$

(never claim)

**Intersection Büchi automaton**

**Background**

- Introduction
- Algorithmic Proposal
- Experiments
- Conclusions & Future Work

**Explicit State MC**

- State Explosion
- Heuristic MC
- Safety & Liveness
- SCCs

**GECCO 2008, Atlanta, USA, July 12-16, 2008**
• **Objective:** Prove that model \( M \) satisfies the property \( f: M \models f \)

• **HSF-SPIN:** the property \( f \) is an LTL formula
Explicit State Model Checking

- **Objective**: Prove that model $M$ satisfies the property $f$: $M \models f$
- **HSF-SPIN**: the property $f$ is an LTL formula

Model $M$

LTL formula $\neg f$

(never claim)

Intersection Büchi automaton

Using Nested-DFS
State Explosion Problem

• Number of states very large even for small models

• Example: Dining philosophers with \( n \) philosophers \( \rightarrow 3^n \) states
  20 philosophers \( \rightarrow 1039 \) GB for storing the states

• Solutions: collapse compression, minimized automaton representation, bitstate hashing, partial order reduction, symmetry reduction

• Large models cannot be verified but errors can be found
The search for errors can be directed by using heuristic information.

Different kinds of heuristic functions have been proposed in the past:

- **Formula-based heuristics**
- **Structural heuristics**
- **Deadlock-detection heuristics**
- **State-dependent heuristics**
Safety and Liveness Properties

Safety property

\[ \forall \sigma \in S^\omega : \sigma \not\in P \Rightarrow (\exists i \geq 0 : \forall \beta \in S^\omega : \sigma_i \beta \not\in P) \]

- Counterexample \(\equiv\) path to accepting state
- Graph exploration algorithms can be used: DFS and BFS

Liveness property

\[ \forall \alpha \in S^* : \exists \beta \in S^\omega, \alpha \beta \vdash P \]

- Counterexample \(\equiv\) path to accepting cycle
- It is not possible to apply DFS or BFS
Strongly Connected Components
Strongly Connected Components
Strongly Connected Components

Concurrent system

\[
\begin{array}{c}
\text{1} \\
\{p\} \\
\{p\} \\
\{p, q\} \\
\text{4} \\
\end{array}
\]

Never claim

\[
\begin{array}{c}
\text{a} \\
\text{true} \\
\neg q \\
\neg q \land p \\
\text{b}
\end{array}
\]

Synchronous product

\[
\begin{array}{c}
\text{1.a} \\
\text{1.b} \\
\text{2.a} \\
\text{2.b} \\
\text{3.a} \\
\text{3.b} \\
\text{4.a} \\
\text{4.b}
\end{array}
\]

GECO 2008, Atlanta, USA, July 12-16, 2008
Strongly Connected Components

- **Introduction**
- **Background**
- **Algorithmic Proposal**
- **Experiments**
- **Conclusions & Future Work**

**Explicit State MC**

- State Explosion
- Heuristic MC
- Safety & Liveness
- SCCs

**Strongly Connected Components**

- Concurrent system
- Never claim
- Synchronous product

- N-SCC
- P-SCC
- F-SCC
Optimization/Search Techniques

- **EXACT**
  - Based on Calculus
    - Newton
    - Gradient
  - Enumeratives
    - Depth First Search
    - Branch and Bound
  - Trayectory-based
    - SA
    - VNS
    - TS

- **APPROXIMATED**
  - Ad Hoc Heuristics
  - METAHEURISTICS
    - EA
    - ACO
    - PSO

Metaheuristics: ACO, ACOhg, ACOhg-live
Ant Colony Optimization (ACO) metaheuristic is inspired by the foraging behaviour of real ants.

ACO Pseudo-code

```plaintext
procedure ACOMetaheuristic
    ScheduleActivities
        ConstructAntsSolutions
        UpdatePheromones
    DaemonActions // optional
end ScheduleActivities
end procedure
```
• The ant selects its next node stochastically

• The probability of selecting one node depends on the pheromone trail and the heuristic value (optional) of the edge/node

• The ant stops when a complete solution is built
ACO: Pheromone Update

- **Pheromone update**
  
  - **During the construction phase**
    \[ \tau_{ij} \leftarrow (1 - \xi) \tau_{ij} \text{ with } 0 \leq \xi \leq 1 \]
  
  - **After the construction phase**
    \[ \tau_{ij} \leftarrow (1 - \rho) \tau_{ij} + \Delta \tau_{ij}^{bs} \text{ with } 0 \leq \rho \leq 1 \]

- **Trail limits (particular of MMAS)**
  
  - Pheromones are kept in the interval \([\tau_{\text{min}}, \tau_{\text{max}}]\)
    
    \[ \tau_{\text{max}} = \frac{Q}{\rho} \]
    \[ \tau_{\text{min}} = \frac{\tau_{\text{max}}}{a} \]
ACOhg: Huge Graphs Exploration

The length of the ant paths is limited by $\lambda_{\text{ant}}$

Initial node

Starting nodes for path construction change

After $\sigma_s$ steps

Second stage

Third stage

What if…?

Objective node

ACOhg: Huge Graphs Exploration

The length of the ant paths is limited by $\lambda_{\text{ant}}$

Initial node

Starting nodes for path construction change

After $\sigma_s$ steps

Second stage

Third stage

What if…?

Objective node
ACOhg-live

- The search is an alternation of two phases

  - **First phase**: search for accepting states
  - **Second phase**: search for cycles from the accepting states

ACOhg-live Pseudocode

```
1: repeat
2:   accept = acohg1.findAcceptingStates(); {First phase}
3:     for node in accept do
4:       acohg2.findCycle(node); {Second phase}
5:     if acohg2.cycleFound() then
6:       return acohg2.acceptingPath();
7:     end if
8:   end for
9:   acohg1.insertTabu(accept);
10: until empty(accept)
11: return null;
```

First phase
The search is an alternation of two phases

- First phase: search for accepting states
- Second phase: search for cycles from the accepting states

**ACOhg-live Pseudocode**

1: repeat
2:  acct = acohg1.findAcceptingStates(); {First phase}
3:  for node in acct do
4:      acohg2.findCycle(node); {Second phase}
5:      if acohg2.cycleFound() then
6:         return acohg2.acceptingPath();
7:      end if
8:  end for
9:  acohg1.insertTabu(acct);
10: until empty(acct)
11: return null;
The search is an alternation of two phases

- **First phase:** search for accepting states
- **Second phase:** search for cycles from the accepting states

ACOhg-live Pseudocode

```
1: repeat
2:     acct = acohg1.findAcceptingStates(); {First phase}
3:     for node in acct do
4:         acohg2.findCycle(node); {Second phase}
5:     if acohg2.cycleFound() then
6:         return acohg2.acceptingPath();
7:     end if
8:     end for
9:     acohg1.insertTabu(acct);
10: until empty(acct)
11: return null;
```
ACOhg-live

• The search is an alternation of two phases

  ➢ First phase: search for accepting states
  ➢ Second phase: search for cycles from the accepting states

ACOhg-live Pseudocode

```
1: repeat
2:   accpt = acohg1.findAcceptingStates(); {First phase}
3:   for node in accpt do
4:     acohg2.findCycle(node); {Second phase}
5:     if acohg2.cycleFound() then
6:       return acohg2.acceptingPath();
7:     end if
8:   end for
9:   acohg1.insertTabu(accpt);
10: until empty(accpt)
11: return null;
```

• Improvement using SCCs

  ➢ First phase: accepting states in N-SCC are ignored
  ➢ Both phases: cycle detected in F-SCC is a violation
Promela Models

- We used 11 Promela models for the experiments

<table>
<thead>
<tr>
<th>Model</th>
<th>LoC</th>
<th>Scalable</th>
<th>Processes</th>
<th>LTL formula (liveness)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alter</td>
<td>64</td>
<td>no</td>
<td>2</td>
<td>□(p → ◊q) ^ □(r → ◊s)</td>
</tr>
<tr>
<td>giopj</td>
<td>740</td>
<td>yes</td>
<td>j+6</td>
<td>□(p → ◊q)</td>
</tr>
<tr>
<td>phi j</td>
<td>57</td>
<td>yes</td>
<td>j+1</td>
<td>□(p → ◊q)</td>
</tr>
<tr>
<td>elevj</td>
<td>191</td>
<td>yes</td>
<td>j+3</td>
<td>□(p → ◊q)</td>
</tr>
<tr>
<td>sgc</td>
<td>1001</td>
<td>no</td>
<td>20</td>
<td>◊p</td>
</tr>
</tbody>
</table>

- Parameters for ACOhg-live

<table>
<thead>
<tr>
<th>Parameter</th>
<th>msteps</th>
<th>cols size</th>
<th>λ_ant</th>
<th>σ_s</th>
<th>ξ</th>
<th>a</th>
<th>ρ</th>
<th>α</th>
<th>β</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st phase</td>
<td>100</td>
<td>10</td>
<td>20</td>
<td>4</td>
<td>0.7</td>
<td>5</td>
<td>0.2</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>2nd phase</td>
<td>20</td>
<td>4</td>
<td></td>
<td></td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Formula-based and finite state machine heuristics
- ACOhg-live implemented in HSF-SPIN
- 100 independent executions and statistical validation
We used 11 Promela models for the experiments

<table>
<thead>
<tr>
<th>Model</th>
<th>LoC</th>
<th>Scalable</th>
<th>Processes</th>
<th>LTL formula (liveness)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alter</td>
<td>64</td>
<td></td>
<td>j=10,15,20</td>
<td>□(p → ◊q) ∧ □(r → ◊s)</td>
</tr>
<tr>
<td>giopj</td>
<td>740</td>
<td>yes</td>
<td>j+6</td>
<td>□(p → ◊q)</td>
</tr>
<tr>
<td>phi j</td>
<td>57</td>
<td>j=20,30,40</td>
<td>j+1</td>
<td>□(p → ◊q)</td>
</tr>
<tr>
<td>elev j</td>
<td>191</td>
<td>yes</td>
<td>j+3</td>
<td>□(p → ◊q)</td>
</tr>
<tr>
<td>sgc</td>
<td>1001</td>
<td></td>
<td>20</td>
<td>◊p</td>
</tr>
</tbody>
</table>

Parameters for ACOhg-live

<table>
<thead>
<tr>
<th>Parameter</th>
<th>msteps</th>
<th>colsize</th>
<th>λ_{ant}</th>
<th>σ_\text{s}</th>
<th>ξ</th>
<th>a</th>
<th>ρ</th>
<th>α</th>
<th>β</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st phase</td>
<td>100</td>
<td>10</td>
<td>20</td>
<td>4</td>
<td>0.7</td>
<td>5</td>
<td>0.2</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>2nd phase</td>
<td>100</td>
<td>20</td>
<td>4</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Formula-based and finite state machine heuristics
- ACOhg-live implemented in HSF-SPIN
- 100 independent executions and statistical validation
Results I: Influence of the SCC Improvement

![Bar chart showing the influence of the SCC improvement with SD marks for select groups.](chart.png)
Results I: Influence of the SCC Improvement

![Bar chart showing the influence of the SCC Improvement]

- Models & parameters
- Results
- Discussion
Results I: Influence of the SCC Improvement

![Graph showing memory usage for different models and parameters with notable results marked as 'SD'].

- Models & parameters
- Results
- Discussion
Results I: Influence of the SCC Improvement

![Bar chart showing the influence of SCC improvement on various models and parameters.](chart.png)
Results II: ACOhg-live+ vs. NDFS and INDFS

<table>
<thead>
<tr>
<th>Models</th>
<th>ACOhg-live+</th>
<th>NDFS</th>
<th>INDFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>alter</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>giop10</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>giop15</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>giop20</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>phi20</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>phi30</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>phi40</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>elev10</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>elev15</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>elev20</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>sgc</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Results II: ACOhg-live$^+$ vs. NDFS and INDFS

Models & parameters

Results

Discussion
Results II: ACOhg-live\(^+\) vs. NDFS and INDFS

### Models & parameters

- ACOhg-live\(^+\)
- NDFS
- INDFS

### Results

<table>
<thead>
<tr>
<th>Model</th>
<th>SD</th>
<th>Memory (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alter</td>
<td>392192</td>
<td>107620</td>
</tr>
<tr>
<td>phi20</td>
<td>388096</td>
<td>15360</td>
</tr>
<tr>
<td>elev10</td>
<td></td>
<td>10001</td>
</tr>
<tr>
<td>elev15</td>
<td></td>
<td>23456</td>
</tr>
<tr>
<td>elev20</td>
<td></td>
<td>34567</td>
</tr>
<tr>
<td>sgc</td>
<td></td>
<td>45678</td>
</tr>
</tbody>
</table>

### Discussion

- The results show significant differences between the algorithms, with ACOhg-live\(^+\) performing better in most cases.
- Further analysis is required to understand the underlying reasons for these differences.

### Conclusions & Future Work

- Further experiments are needed to validate the findings.
- Consider exploring the integration of additional parameters to improve performance.
Results II: ACOhg-live$^+$ vs. NDFS and INDFS

<table>
<thead>
<tr>
<th>Models &amp; parameters</th>
<th>Results</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOhg-live$^+$</td>
<td>NDFS</td>
<td>INDFS</td>
</tr>
<tr>
<td>alter</td>
<td>phi20</td>
<td>elev10</td>
</tr>
<tr>
<td>10001</td>
<td>1069</td>
<td>9999</td>
</tr>
<tr>
<td>SD</td>
<td>392192</td>
<td>388096</td>
</tr>
<tr>
<td>15670</td>
<td>15320</td>
<td>1059</td>
</tr>
<tr>
<td>sgc</td>
<td>10001</td>
<td>15320</td>
</tr>
</tbody>
</table>
How to use ACOhg-live

• ACOhg-live should be used in the first/middle stages of the software development, when software errors are expected

• ACOhg-live can also be used in other phases of the software development for testing concurrent software
Conclusions

- ACOhg-live is the first algorithm based on metaheuristics (to the best of our knowledge) applied to the search for liveness errors in concurrent models
- The improvement based on the SCCs of the never claim outperforms the efficacy of ACOhg-live
- ACOhg-live is able to outperform (Improved) Nested-DFS in efficacy and efficiency in the search for liveness errors

Future Work

- Analysis of parameterization for reducing the parameters
- Include ACOhg-live into JavaPathFinder for finding liveness errors in Java programs
- Combine ACOhg-live with techniques for reducing the memory required for the search such as partial order reduction (work in progress)
Searching for Liveness Property Violations in Concurrent Systems with ACO

Thanks for your attention !!!

GECCO 2008, Atlanta, USA, July 12-16, 2008