Comparison of Algorithms for Checking Emptiness of Büchi Automata

Andreas Gaiser¹, Stefan Schwoon²

¹Fakultät für Informatik, Technische Universität München, Germany
²LSV, CNRS, ENS de Cachan, INRIA Saclay, France

Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, Znojmo, 2009
Emptiness checks in LTL Model Checking

The Testbed

Emptiness Algorithms
- Algorithms based on Nested Depth First Search
  - Holzmann/Peled/Yannakakis (HPY)
  - Amended Nested Depth First Search (AND)
- SCC-based algorithms
  - Tarjan’s algorithm
  - Geldenhuys/Valmari (GV)
  - Couvreur’s Algorithm (C99)
  - Amended Couvreur’s Algorithm (ASCC)

Conclusion
Finite-state Model Checking using LTL

- Given:
  - Formal description of a finite-state system $S$
  - An LTL formula $\phi$ representing a specification property
- Does $S \models \phi$ hold?
- $S$ and $\phi$ are used to construct a Büchi automaton $A$
- $S \models \phi$ iff $L(A) = \emptyset$ (Emptiness problem)
Satisfiability check

- $\mathcal{L}(A) \neq \emptyset$ can be reduced to a graph-theoretic problem

$\Rightarrow$ Find a lasso in the transition diagram of $A$

- Lasso: Cycle containing an accepting state which is reachable from initial state
Satisfiability check

- $\mathcal{L}(A) \neq \emptyset$ can be reduced to a graph-theoretic problem
- Find a lasso in the transition diagram of $A$
- Lasso: Cycle containing an accepting state which is reachable from initial state
Model Checking on-the-fly

- State space of $\mathcal{A}$ might be huge

⇒ Lassos might be found without constructing the entire automaton $\mathcal{A}$

- On-the-fly algorithms for finding a lasso
  - Start with the initial state of the automaton $\mathcal{A}$
  - Use function $\text{post}(s)$: Computes all direct successor states of $s$ ("Generate-by-need")
  - Store state descriptors + additional information for every generated state (no need for storing transitions)

\[
\begin{array}{c}
\text{...}
\end{array}
\]
Our contribution

- Several on-the-fly algorithms for the emptiness problem linear in the size of $\mathcal{A}$ do exist
- Do they take good advantage of the on-the-fly approach?
  ⇒ Experimental comparison of algorithms with regard to speed and memory consumption:
    - Nested Depth First Search variants ("classical" approach)
    - SCC-based approaches
  ⇒ Development of two amended algorithms, AND and ASCC, based on our observations.
Our Testbed - The Ingredients

- Implementation based on the virtual machine NIPS\(^1\)
- NIPS library used as state space generator (\textit{post}-function)
- Promela Compiler: Transforms Promela-programs written for Spin into NIPS-bytecode

\(^1\)http://wwwhome.cs.utwente.nl/michaelw/nips/
Our Testbed

- 16 parametrizable models from the BEEM$^2$ library, e.g. ethernet protocols, distributed algorithms, ...
- 266 test cases (combinations of formulas and model instances)
- All algorithms tested within the same framework
- Setting similar to “classical” Spin:
  - All generated states explicitly stored, no use of symbolic representations (e.g. BDDs)
  - Sequential methods, no parallelization techniques
  - No approximative methods, e.g. bitstate hashing

$^2$http://anna.fi.muni.cz/models/
Important Experimental Observations

- **Number of explored states**: Dominating factor of space consumption
- **Number of post-calls**: Dominating factor of running time
- Previous tests and experiments often use preconstructed Büchi automata ⇒ Cost for post not properly taken into account
1 Emptiness checks in LTL Model Checking

2 The Testbed

3 Emptiness Algorithms
   - Algorithms based on Nested Depth First Search
     - Holzmann/Peled/Yannakakis (HPY)
     - Amended Nested Depth First Search (AND)
   - SCC-based algorithms
     - Tarjan’s algorithm
     - Geldenhuys/Valmari (GV)
     - Couvreur’s Algorithm (C99)
     - Amended Couvreur’s Algorithm (ASCC)

4 Conclusion
“Classical” Nested DFS: HPY
(Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS

A. Gaiser, S. Schwoon (TUM, LSV)
Checking Emptiness of Büchi Automata
MEMICS 2009 11 / 21
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS

Diagram:

- Nodes: 1, 2, 3, 4, 5, 6
- Edges:
  - 1 to 2
  - 2 to 3
  - 3 to 4
  - 4 to 3
  - 3 to 5
  - 5 to 6
  - 6 back to 3

Process:
- (2) has already been visited by the blue search, backtrack to (4), (3)
- Backtrack to (5)
- Red DFS starting from (5)
- Red DFS terminated, backtrack to (3), (2)
- Red DFS starting from (2)
- Path found from (3) to (3)
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a **blue** DFS procedure and a (nested) **red** DFS procedure
- **Red** DFS sets **blue** states to **red**
- Starts with **blue** DFS
“Classical” Nested DFS: HPY
(Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
- (2) has already been visited by the blue search, backtrack to (4), (3)
“Classical” Nested DFS: HPY
(Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
- (2) has already been visited by the blue search, backtrack to (4), (3)
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
- (2) has already been visited by the blue search, backtrack to (4), (3)

\[ \text{Diagram showing the DFS traversal steps.} \]
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a **blue** DFS procedure and a (nested) **red** DFS procedure
- **Red** DFS sets **blue** states to **red**
- Starts with **blue** DFS
- (2) has already been visited by the **blue** search, backtrack to (4), (3)
- Backtrack to (5)

![Diagram](image-url)
“Classical” Nested DFS: HPY
(Holzmann/Peled/Yannakakis)

- Uses a **blue** DFS procedure and a (nested) **red** DFS procedure
- **Red** DFS sets **blue** states to **red**
- Starts with **blue** DFS
- (2) has already been visited by the **blue** search, backtrack to (4), (3)
- Backtrack to (5)
- **Red** DFS starting from (5)
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
- (2) has already been visited by the blue search, backtrack to (4), (3)
- Backtrack to (5)
- Red DFS starting from (5)
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a **blue** DFS procedure and a (nested) **red** DFS procedure
- **Red** DFS sets **blue** states to **red**
- Starts with **blue** DFS
- (2) has already been visited by the **blue** search, backtrack to (4), (3)
- Backtrack to (5)
- **Red** DFS starting from (5)
- **Red** DFS terminated, backtrack to (3), (2)
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
- (2) has already been visited by the blue search, backtrack to (4), (3)
- Backtrack to (5)
- Red DFS starting from (5)
- Red DFS terminated, backtrack to (3), (2)
- Red DFS starting from (2)

Diagram:

1. Blue state
2. Red state
3. Blue state
4. Blue state
5. Red state
6. Blue state...

A. Gaiser, S. Schwoon (TUM, LSV) Checking Emptiness of Büchi Automata MEMICS 2009 11 / 21
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
- (2) has already been visited by the blue search, backtrack to (4), (3)
- Backtrack to (5)
- Red DFS starting from (5)
- Red DFS terminated, backtrack to (3), (2)
- Red DFS starting from (2)
“Classical” Nested DFS: HPY
(Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
- (2) has already been visited by the blue search, backtrack to (4), (3)
- Backtrack to (5)
- Red DFS starting from (5)
- Red DFS terminated, backtrack to (3), (2)
- Red DFS starting from (2)
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
- (2) has already been visited by the blue search, backtrack to (4), (3)
- Backtrack to (5)
- Red DFS starting from (5)
- Red DFS terminated, backtrack to (3), (2)
- Red DFS starting from (2)
“Classical” Nested DFS: HPY (Holzmann/Peled/Yannakakis)

- Uses a blue DFS procedure and a (nested) red DFS procedure
- Red DFS sets blue states to red
- Starts with blue DFS
- (2) has already been visited by the blue search, backtrack to (4), (3)
- Backtrack to (5)
- Red DFS starting from (5)
- Red DFS terminated, backtrack to (3), (2)
- Red DFS starting from (2)
- Path found from (3) to (3)
Amended Nested Depth First Search (AND)

- Based on Four Colour algorithm (Schwoon/Esparza): Improved version of HPY
- Frequently occurring pattern in investigated Büchi automata:
  - Many accepting states not within any cycle
  - Unsuccessful red search invokes post-calls!
- Amended Nested DFS (AND):
  Checks whether all successors of an accepting state \( q \) have become red \( \Rightarrow \) Improvement in 86 test cases

(Detail from a model of Lamport's algorithm)
The Trouble with Nested-DFS Algorithms

- Widely used, e.g. HPY in Spin
- Need only 2-3 additional bits per state
- But: Explores additional states even though all transitions of a lasso have already been discovered!
  - **Number of explored states**: Dominating factor of space consumption
  - **Number of post-calls**: Dominating factor of running time

![Diagram showing a large subgraph without accepting states]
SCC-based algorithms

- Not widely used yet
- Main Idea: Find strongly connected components (SCCs) reachable from the initial state using Tarjan’s algorithm
- If a nontrivial SCC contains an accepting state ⇒ Lasso is found
SCC-based algorithms

- Not widely used yet
- Main Idea: Find strongly connected components (SCCs) reachable from the initial state using Tarjan’s algorithm
- If a nontrivial SCC contains an accepting state ⇒ Lasso is found
SCC-based algorithms

- Not widely used yet
- Main Idea: Find strongly connected components (SCCs) reachable from the initial state using Tarjan’s algorithm
- If a nontrivial SCC contains an accepting state ⇒ Lasso is found
Tarjan’s algorithm

- DFS-based approach
- Every newly explored state is inserted in Current-Stack
- After exploring all states of SCC $S$: All states of $S$ are located on top of the Current-Stack
- Remove them, thereby check for accepting state
- Entire SCC $S$ has to be explored before lasso check $\Rightarrow$ No On-the-fly possible!
SCC-based approaches: Geldenhuys/Valmari (GV) and Couvreur’s algorithm (C99)

- **Idea:** Insert Checks for lassos before generating the entire SCC!
- **Detect a cycle as soon as all its transitions and states are explored** ⇒ **Minimal exploration!**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Needs (per state)</th>
<th>Minimal exploration?</th>
<th>Maximal post-calls per state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarjan</td>
<td>2 integer + 1 bit</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>GV</td>
<td>2 integer + 1 bit</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>C99</td>
<td>1 integer + 1 bit</td>
<td>yes</td>
<td>2</td>
</tr>
</tbody>
</table>

Advantage over other algorithms: C99 can be extended to handle generalized Büchi automata

C99 favoured in previous publications (Schwoon/Esparza 2005)

A. Gaiser, S. Schwoon (TUM, LSV) — Checking Emptiness of Büchi Automata — MEMICS 2009
SCC-based approaches: Geldenhuys/Valmari (GV) and Couvreur’s algorithm (C99)

- Idea: Insert Checks for lassos before generating the entire SCC!
- Detect a cycle as soon as all its transitions and states are explored ⇒ Minimal exploration!

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Needs (per state)</th>
<th>Minimal exploration?</th>
<th>Maximal post-calls per state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarjan</td>
<td>2 integer + 1 bit</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>GV</td>
<td>2 integer + 1 bit</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>C99</td>
<td>1 integer + 1 bit</td>
<td>yes</td>
<td>2</td>
</tr>
</tbody>
</table>

- Advantage over other algorithms: C99 can be extended to handle generalized Büchi automata
- C99 favoured in previous publications (Schwoon/Esparza 2005)
Couvreur’s Algorithm (C99)

- Current represented as bit per state, not as Stack
- After detecting an SCC $S$, secondary DFS is started to update Current
  \[ \Rightarrow \text{Needs two calls of } post(s) \text{ for every state in } S \]
- SCCs without accepting states in 98 test cases
- Examples with no lasso \[ \Rightarrow \text{C99 twice as long as GV or Nested DFS!} \]
Amended Couvreur’s Algorithm (ASCC)

- Idea: Reinsert the Current-stack in C99
- Second DFS can be replaced by removing elements from the Current-Stack

⇒ Only one post-call per state!

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Needs (per state)</th>
<th>Minimal exploration?</th>
<th>max. post-calls per state</th>
<th>GBA?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarjan</td>
<td>2 integer + 1 bit</td>
<td>no</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>GV</td>
<td>2 integer + 1 bit</td>
<td>yes</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>C99</td>
<td>1 integer + 1 bit</td>
<td>yes</td>
<td>2</td>
<td>yes</td>
</tr>
<tr>
<td>ACC</td>
<td>1 integer + 1 bit</td>
<td>yes</td>
<td>1</td>
<td>yes</td>
</tr>
</tbody>
</table>
Comparison - Running times

- Running times (ordering implementation-independent):

<table>
<thead>
<tr>
<th>algorithm</th>
<th>run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCC</td>
<td>67.0 %</td>
</tr>
<tr>
<td>Geldenhuys/Valmari (GV)</td>
<td>69.2 %</td>
</tr>
<tr>
<td>AND</td>
<td>69.7 %</td>
</tr>
<tr>
<td>Schwoon/Esparza (SE)</td>
<td>96.3 %</td>
</tr>
<tr>
<td>Holzmann/Peled/Yannakakis (HPY)</td>
<td>100.0 %</td>
</tr>
<tr>
<td>Couvreur (C99)</td>
<td>128.3 %</td>
</tr>
</tbody>
</table>

- Size of state descriptors: 20 - 380 bytes, 130 bytes on average
- Additional space per state (max. 2 ints + 1 bit) negligible

⇒ Number of explored states: Dominating factor of space consumption
- Observation: Nested DFS algorithms ran out of memory more often than SCC-based algorithms
Conclusion: Our Recommendation 😊

- **ASCC** or **GV** for explicit on-the-fly model checking with LTL and simple Büchi automata
- **ASCC** if generalized Büchi automata are processed
- **AND** if certain techniques (e.g. bitstate hashing) are used
- There remains no reason to use SE, HPY or C99 anymore.
The End.

Thank you!

Stefan Schwoon, Javier Esparza: *A note on on-the-fly verification algorithms*, In TACAS, pages 174-190, 2005

