Improved On-The-Fly Livelock Detection with $\text{DFS}_{\text{FIFO}}$

Alfons Laarman

Joint with David Faragó
(Karlsruhe Institute of Technology)

May 14th, 2013

NFM, Moffett Field, CA, USA
Dealing with State Space Explosion

Problem
State Explosion in LTL Model Checking

Viable solutions:

- Parallelization
  ⇕
- Partial-Order/Confluence Reduction
  ⇕
- Specialization on subclasses of LTL (Livelocks, Weak LTL)

(LaQuSo project)
<table>
<thead>
<tr>
<th></th>
<th>1. Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>LTL</td>
</tr>
<tr>
<td>3</td>
<td>Parallelism</td>
</tr>
<tr>
<td>4</td>
<td>Partial-Order Reduction</td>
</tr>
<tr>
<td>5</td>
<td>DFS_{FIFO}</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
LTL Model Checking

\[ M : \]

\[ \downarrow \]

\[ TS : \]

\( A \phi \) (Büchi):

\( b \sim a, b \sim 2 \)

\( LTL \) formula, e.g.:

\( \phi = \Box(a \Rightarrow \Diamond b) \)
LTL Model Checking

\[ M \models \varphi \]

LTL formula, e.g.:
\[ \varphi = \square(a \Rightarrow \Diamond b) \]
LTL Model Checking

$\mathcal{M} = \varphi$

$L(\mathcal{T}S) \subseteq L(A\varphi)$

LTL formula, e.g.:

$\varphi = \Box(a \Rightarrow \Diamond b)$

$A\varphi$ (Büchi):

$\neg a, b$
LTL Model Checking

\[ M \models \varphi \]
\[ L(TS) \subseteq L(A_\varphi) \]
\[ L(TS) \cap L(A_{\neg \varphi}) = \emptyset \]

LTL formula, e.g.:
\[ \varphi = \square(a \Rightarrow \diamond b) \]

\[ A_\varphi \text{ (Büchi)} : \]
\[ \neg a, b \]
LTL Model Checking

\[ \mathcal{M} \models \varphi \]
\[ \iff \]
\[ \mathcal{L}(\mathcal{T}S) \subseteq \mathcal{L}(A_{\varphi}) \]
\[ \iff \]
\[ \mathcal{L}(\mathcal{T}S) \cap \mathcal{L}(A_{\neg \varphi}) = \emptyset \]
\[ \iff \]
\[ \mathcal{T}S \otimes A_{\neg \varphi} \]
contains no accepting cycle

LTL formula, e.g.:
\[ \varphi = \Box (a \Rightarrow \Diamond b) \]

\[ \iff \]
\[ A_{\varphi} \text{ (Büchi)} : \]
\[ \neg a, b \]
**LTL Model Checking**

\[ M \models \varphi \]

\[ L(TS) \subseteq L(A\varphi) \]

\[ L(TS) \cap L(A\neg\varphi) = \emptyset \]

\[ TS \otimes A\neg\varphi \]

contains no accepting cycle

LTL formula, e.g.:

\[ \varphi = \square(a \Rightarrow \Diamond b) \]

\[ A\varphi \text{ (Büchi): } \neg a, b \]
LTL Model Checking

\[ \mathcal{M} \models \varphi \]

\[ \mathcal{L}(TS) \subseteq \mathcal{L}(A_\varphi) \]

\[ \mathcal{L}(TS) \cap \mathcal{L}(A_{\neg \varphi}) = \emptyset \]

\[ TS \otimes A_{\neg \varphi} \]

contains no accepting cycle

LTL formula, e.g.:

\[ \varphi = \Box(a \Rightarrow \Diamond b) \]

C:

\[ 2^C \times 2^{AP} \]

Improved On-The-Fly Livelock Detection with DFSFIFO
Algorithm

- Two DFS searches
- Linear \((2N)\) in size of the cross product
- DFS order (hard to parallelize)
CNDFS for Parallel LTL Model Checking

Simple idea: *P independent, randomized* NDFS instances (swarm)
Store states in lockless hash/tree table [FMCAD 2010/SPIN 2011]

- More on the fly (bug hunting)
- No speedup for full verification
CNDFS for Parallel LTL Model Checking

Simple idea: $P$ independent, randomized NDFS instances (swarm)
Store states in lockless hash/tree table [FMCAD 2010/SPIN 2011]

+ More on the fly (bug hunting)
— No speedup for full verification

Better idea: Store information in graph to prune work:
Multi-core NDFS $\rightarrow$ CNDFS [ATVA 2011/2012]
CNDFS for Parallel LTL Model Checking

Simple idea: *P independent, randomized* NDFS instances (swarm)
Store states in lockless hash/tree table [FMCAD 2010/SPIN 2011]

+ More on the fly (bug hunting)

— No speedup for full verification

Better idea: Store information in graph to prune work:
Multi-core NDFS $\rightarrow$ CNDFS [ATVA 2011/2012]
CNDFS for Parallel LTL Model Checking

Simple idea: $P$ independent, randomized NDFS instances (swarm)
Store states in lockless hash/tree table [FMCAD 2010/SPIN 2011]

+ More on the fly (bug hunting)
— No speedup for full verification

Better idea: Store information in graph to prune work:
Multi-core NDFS $\rightarrow$ CNDFS [ATVA 2011/2012]

Worst case complexity: $N \times P$, but in practice close to $N$
LTL property: $\square$\textit{progress}

<table>
<thead>
<tr>
<th></th>
<th>SPIN NDFS</th>
<th>LTSmin CNDFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>48 cores</td>
<td></td>
</tr>
<tr>
<td>$?extleader_t$</td>
<td>1390</td>
<td>926</td>
</tr>
<tr>
<td>garp</td>
<td>2050</td>
<td>1061</td>
</tr>
<tr>
<td>i-prot</td>
<td>103</td>
<td>76</td>
</tr>
</tbody>
</table>
LTL property: $\Box \Diamond progress$

<table>
<thead>
<tr>
<th>Protocol</th>
<th>SPIN NDFS</th>
<th>LTSmin NDFS</th>
<th>CNDFS 1 core</th>
<th>CNDFS 48 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>extleader_t</td>
<td>1390</td>
<td>926</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>garp</td>
<td>2050</td>
<td>1061</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>i-prot</td>
<td>103</td>
<td>76</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Graph shows speedup over threads for different protocols:
- cndfs
- garp
- giop2.nomig
- i-protocol2
- leader5
Partial-Order Reduction

\( \mathcal{T}S : \)
Partial-Order Reduction

$\mathcal{T}_S : \quad \text{deadlock} \quad \rightarrow \quad \mathcal{T}_S :$

UNIVERSITY OF TWENTE.
Partial-Order Reduction

$\mathcal{T}S:\hspace{1cm} \text{deadlock} \hspace{1cm} \mathcal{T}S:\hspace{1cm} +\text{cycle}$

$\downarrow$
Partial-Order Reduction

\[ \mathcal{T}S : \]

\[ \mathcal{T}S : \]

\[ +\text{visibility} \leftarrow \]

\[ +\text{cycle} \downarrow \]

\[ +\text{visibility} \leftarrow \]

\[ +\text{cycle} \downarrow \]
Partial-Order Reduction for (C)NDFS

- stack proviso for NDFS
Partial-Order Reduction for (C)NDFS

- stack proviso for NDFS
- CNDFS has incomplete stacks $\Rightarrow$ No POR
Partial-Order Reduction for (C)NDFS

- stack proviso for NDFS
- CNDFS has incomplete stacks $\Rightarrow$ No POR

Other solutions to solve the problem of parallelism & POR:
OWCTY + Topological Sort Proviso [Barnat et al. 2010]
Specializing on Livelock Detection

Livelock LTL property: $\Box \Diamond P$

Livelocks are important properties, they are used for:

- $\frac{1}{3}$ of all the BEEM models
- $\frac{1}{2}$ of the models in the Promela Database
  (http://www.albertolluch.com/research/promelamodels)
Livelock LTL property: \( \Box \Diamond P \)

Livelocks are important properties, they are used for:

- \( \frac{1}{3} \) of all the BEEM models
- \( \frac{1}{2} \) of the models in the Promela Database
  (http://www.albertolluch.com/research/promelamodels)
- Progress detection DFS [Holzmann et al. 1996]
- DFS\(_{FIFO} \) with Partial-Order Reduction [Farago et al. 2009]
Specializing on Livelock Detection

Livelock LTL property: $\square \Diamond P$

Livelocks are important properties, they are used for:

- $\frac{1}{3}$ of all the BEEM models
- $\frac{1}{2}$ of the models in the Promela Database
  (http://www.albertolluch.com/research/promelamodels)
- Progress detection DFS [Holzmann et al. 1996]
- $\text{DFS}_{\text{FIFO}}$ with Partial-Order Reduction [Farago et al. 2009]

$\mathcal{T}S : P$
DFS\_FIFO for Livelock Detection

\[ \rightarrow P \quad \neg P \quad \rightarrow P \quad \rightarrow P \quad \rightarrow P \]

DFS    BFS
DFS\textsubscript{FIFO} for Liveloock Detection
**DFS** for Livello Detection
DFS\textsubscript{FIFO} for Livelock Detection
Expected outcome:

- $\text{DFS}_{\text{FIFO}}$ makes single pass over state space (unlike NDFS)
- $|\mathcal{T}S| \leq \frac{1}{2} |\mathcal{C}|$

Up to 4 times as fast as NDFS
Expected outcome:

- $\text{DFS}_{\text{FIFO}}$ makes single pass over state space (unlike NDFS)
- $|\mathcal{T}\mathcal{S}| \leq \frac{1}{2}|\mathcal{C}|$

Up to 4 times as fast as NDFS

<table>
<thead>
<tr>
<th></th>
<th>LTSmin (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{DFS}_{\text{FIFO}}$</td>
</tr>
<tr>
<td>garp</td>
<td>591.2</td>
</tr>
<tr>
<td>i-prot</td>
<td>41.4</td>
</tr>
<tr>
<td>$?\text{extleader}_t$</td>
<td>233.2</td>
</tr>
</tbody>
</table>
Cycle proviso

- progress cycle $\Rightarrow$ visibility proviso
- non-progress cycle $\Rightarrow$ report counter example

No additional stack proviso required!
Cycle proviso

- progress cycle $\Rightarrow$ visibility proviso
- non-progress cycle $\Rightarrow$ report counter example

No additional stack proviso required!

<table>
<thead>
<tr>
<th>Model</th>
<th>States</th>
<th>LTSmin (POR)</th>
<th>SPIN (POR)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DFS$_{\text{FIFO}}$</td>
<td>NDFS</td>
</tr>
<tr>
<td>garp</td>
<td>72,318,749</td>
<td>2.2%</td>
<td>32.6%</td>
</tr>
<tr>
<td>i-protocol</td>
<td>20,052,267</td>
<td>31.8%</td>
<td>32.0%</td>
</tr>
<tr>
<td>?extleader$_t$</td>
<td>89,771,572</td>
<td>0.5%</td>
<td>0.5%</td>
</tr>
</tbody>
</table>

LTSmin uses the stubborn set method [SPIN 2013].
DFS_FIFO and Partial-Order Reduction

Cycle proviso

- progress cycle $\Rightarrow$ visibility proviso
- non-progress cycle $\Rightarrow$ report counter example

No additional stack proviso required!

<table>
<thead>
<tr>
<th>Model</th>
<th>States</th>
<th>LTSmin (POR)</th>
<th>SPIN (POR)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DFS_FIFO NDFS</td>
<td>NDFS</td>
</tr>
<tr>
<td>garp</td>
<td>72,318,749</td>
<td>2.2% 32.6%</td>
<td>18.3%</td>
</tr>
<tr>
<td>i-protocol</td>
<td>20,052,267</td>
<td>31.8% 32.0%</td>
<td>41.4%</td>
</tr>
<tr>
<td>$extleader_t$</td>
<td>89,771,572</td>
<td>0.5% 0.5%</td>
<td>1.2%</td>
</tr>
</tbody>
</table>

LTSmin uses the stubborn set method [SPIN 2013]
Parallel DFS_{FIFO}
Introduction

LTL Parallelism

Partial-Order Reduction

DFS FIFO

Conclusion

Parallel DFS FIFO Experiments

No cycle proviso \[\Rightarrow\] Parallelism + POR!

UNIVERSITY OF TWENTE.

Improved On-The-Fly Livelong Detection with DFS FIFO

May 14th, 2013
Parallel DFS\textsubscript{FIFO} Experiments

![Graph](image_url)

No cycle proviso \(\Rightarrow\) Parallelism + POR!
Parallel LTL with Partial Order Reduction

\(\text{DFS}_{\text{FIFO}} \text{ vs OWCTY & Topological Sort Proviso} \)

[Barnat et al 2010]
### Parallel LTL with Partial Order Reduction

**DFS\_FIFO vs OWCTY & Topological Sort Proviso**

[Barnat et al 2010]

| N   | Alg.    | |R| |T| |T_1| |T_{48}| |U| |T^{por}| |R^{por}| |T^{por}| |T^{por}_{1}| |T^{por}_{48}| |U^{por}| |
|-----|---------|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 9   | cndfs   | 3.6E7| 2.3E8| 502.6| 12.0| 41.8|    |    |    |    |    |    |    |    |    |    |
| 9   | pdfs\_fifo | 3.6E7| 2.3E8| 583.6| 14.3| 40.8|    |    |    |    |    |    |    |    |    |    |
| 9   | owcty   | 3.6E7| 2.3E8| 498.7| 51.9| 9.6 |    |    |    |    |    |    |    |    |    |    |
| 10  | cndfs   | 2.4E8| 1.7E9| 30'  | 90.7| 30' |    |    |    |    |    |    |    |    |    |    |
| 10  | pdfs\_fifo | 2.4E8| 1.7E9| 30'  | 109.3| 30' |    |    |    |    |    |    |    |    |    |    |
| 10  | owcty   | 2.4E8| 1.7E9| 30'  | 663.1| 30' |    |    |    |    |    |    |    |    |    |    |
| 11  | pdfs\_fifo | 30'  | 30'  | 30'  | 30'  | 30' |    |    |    |    |    |    |    |    |    |    |
| 11  | owcty   | 30'  | 30'  | 30'  | 30'  | 30' |    |    |    |    |    |    |    |    |    |    |
| 12  | pdfs\_fifo | 30'  | 30'  | 30'  | 30'  | 30' |    |    |    |    |    |    |    |    |    |    |
| 13  | pdfs\_fifo | 30'  | 30'  | 30'  | 30'  | 30' |    |    |    |    |    |    |    |    |    |    |
| 14  | pdfs\_fifo | 30'  | 30'  | 30'  | 30'  | 30' |    |    |    |    |    |    |    |    |    |    |
| 15  | pdfs\_fifo | 30'  | 30'  | 30'  | 30'  | 30' |    |    |    |    |    |    |    |    |    |    |
Conclusions

By specializing for livelocks with DFS\textsubscript{FIFO}, we

- improved partial-order reduction
- improved the efficiency of parallelization by 100%
- allow efficient combination of POR and parallelism
Conclusions

By specializing for livelocks with $\text{DFS}_{\text{FIFO}}$, we

- improved partial-order reduction
- improved the efficiency of parallelization by 100%
- allow efficient combination of POR and parallelism

Future work

- use testers to support more LTL properties [Valmari CAV ’93]
## Conclusions

By specializing for livelocks with $\text{DFS}_{\text{FIFO}}$, we

- improved partial-order reduction
- improved the efficiency of parallelization by 100%
- allow efficient combination of POR and parallelism

## Future work

- use testers to support more LTL properties [Valmari CAV ’93]

## Other mentioned works

1. Boosting Multi-Core Reachability Performance with Shared Hash Tables – FMCAD’10
2. Parallel Recursive State Compression for Free – SPIN’11
3. Multi-Core Nested Depth-First Search – ATVA’11
4. Improved Multi-Core Nested Depth-First Search – ATVA’12
5. SpinS: Extending LTSmin with Promela through SpinJa – PDMC’12