INVARIANTS FOR FINITE INSTANCES AND BEYOND

October, 21st 2013

Sylvain Conchon, Amit Goel, Sava Kristić, Alain Mebsout, Fatiha Zaïdi

LRI, Université Paris-Sud
Strategic CAD Labs, Intel Corporation
How to prove safety of industrial size protocols like FLASH for an arbitrary number of processes?
How to prove safety of industrial size protocols like FLASH for an arbitrary number of processes?

- automatically
The FLASH protocol

Stanford FLASH multiprocessor architecture (1994)

- Cache-coherence shared memory
- High-performance message passing
- Industrial size: **67 million** states for 4 processes (28,000 states for German)

Who proved the protocol?

- Park and Dill, 1996, PVS proof
- Das, Dill and Park, 1999, by predicate abstraction
- McMillan, 2001, by compositional model checking
- Chou, Mannava, Park, 2004, CMP method inspired by McMillan’s work
- Talapur and Tuttle, 2008, message-flows extension of CMP

None of these proofs are purely automatic.
The FLASH protocol

Stanford FLASH multiprocessor architecture (1994)

- Cache-coherence shared memory
- High-performance message passing
- Industrial size: 67 million states for 4 processes (28,000 states for German)

Who proved the protocol?

- Park and Dill, 1996, PVS proof
- Das, Dill and Park, 1999, by predicate abstraction
- McMillan, 2001, by compositional model checking
- Chou, Mannava, Park, 2004, CMP method inspired by McMillan’s work
- Talapur and Tuttle, 2008, message-flows extension of CMP

None of these proofs are purely automatic
- Model checking of parameterized systems
- Decidable fragment
- Cubicle implements backward reachability
Solutions

- Model checking of parameterized systems
- Decidable fragment
- Cubicle implements backward reachability

Does it work?
Some benchmarks

<table>
<thead>
<tr>
<th></th>
<th>Cubicle</th>
<th>CMurphi</th>
<th>CMurphi</th>
<th>CMurphi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Szymanski_at</td>
<td>0.30s</td>
<td>8.04s (8)</td>
<td>5m12s (10)</td>
<td>2h50m (12)</td>
</tr>
<tr>
<td>German_Baukus</td>
<td>7.03s</td>
<td>0.74s (4)</td>
<td>19m35s (8)</td>
<td>4h49m (10)</td>
</tr>
<tr>
<td>German.CTC</td>
<td>3m23s</td>
<td>1.83s (4)</td>
<td>43m46s (8)</td>
<td>12h35m (10)</td>
</tr>
<tr>
<td>German_pfs</td>
<td>3m58s</td>
<td>0.99s (4)</td>
<td>22m56s (8)</td>
<td>5h30m (10)</td>
</tr>
<tr>
<td>Chandra-Toueg</td>
<td>2h01m</td>
<td>5.68s (4)</td>
<td>2m58s (5)</td>
<td>1h36m (6)</td>
</tr>
</tbody>
</table>
## Some benchmarks

<table>
<thead>
<tr>
<th></th>
<th>Cubicle</th>
<th></th>
<th>CMurphi</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Szymanski_at</td>
<td>0.30s</td>
<td>8.04s (8)</td>
<td>5m12s (10)</td>
<td>2h50m (12)</td>
<td></td>
</tr>
<tr>
<td>German_Baukus</td>
<td>7.03s</td>
<td>0.74s (4)</td>
<td>19m35s (8)</td>
<td>4h49m (10)</td>
<td></td>
</tr>
<tr>
<td>German_CTC</td>
<td>3m23s</td>
<td>1.83s (4)</td>
<td>43m46s (8)</td>
<td>12h35m (10)</td>
<td></td>
</tr>
<tr>
<td>German_pfs</td>
<td>3m58s</td>
<td>0.99s (4)</td>
<td>22m56s (8)</td>
<td>5h30m (10)</td>
<td></td>
</tr>
<tr>
<td>Chandra-Toueg</td>
<td>2h01m</td>
<td>5.68s (4)</td>
<td>2m58s (5)</td>
<td>1h36m (6)</td>
<td></td>
</tr>
<tr>
<td>Szymanski_na</td>
<td>T.O.</td>
<td>0.88s (4)</td>
<td>8m25s (6)</td>
<td>7h08m (8)</td>
<td></td>
</tr>
<tr>
<td>Flash_nodata</td>
<td>O.M.</td>
<td>4.86s (3)</td>
<td>3m33s (4)</td>
<td>2h46m (5)</td>
<td></td>
</tr>
<tr>
<td>Flash</td>
<td>O.M.</td>
<td>1m27s (3)</td>
<td>2h15m (4)</td>
<td>O.M. (5)</td>
<td></td>
</tr>
</tbody>
</table>

O.M. > 20 GB

T.O. > 20 h
How to scale?

- Reduce the state space to explore
- Invariants for parameterized case
- Interesting behaviors often observable on small instances
Problem: Invariants often harder to prove than original property
Problem: Invariants often harder to prove than original property

Idea: use finite instances to infer invariants for parametrized case

- Insert and check on the fly in backward reachability loop
- Backtrack if necessary

BRAB: Backward Reachability with Approximations and Backtracking
Backward reachability algorithm
Backward reachability algorithm
Backward reachability algorithm
Backward reachability algorithm
Backward reachability algorithm
Backward reachability algorithm
Backward reachability algorithm
Backward reachability algorithm
Backward reachability algorithm
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
BRAB: intuition
Symbolic framework for parameterized systems

States: formulas in a decidable fragment of FOL

Pre-image effectively computable

Post-image effectively computable for a finite instance
Symbolic framework for parameterized systems

States: formulas in a decidable fragment of FOL

Pre-image effectively computable

Post-image effectively computable for a finite instance

In Cubicle → array-based transition systems
Example: German-**ish** cache coherence protocol

**Client** \(i\):
\[
\text{Cache}[i] \in \{E, S, I\}
\]

**Directory:**
\[
\text{Cmd} \in \{rs, re, \epsilon\}
\]
\[
\text{Ptr} \in proc
\]
\[
\text{Shr}[i] \in \{\text{true}, \text{false}\}
\]
\[
\text{Exg} \in \{\text{true}, \text{false}\}
\]

**Initial states:**
\[
\forall i. \text{Cache}[i] = I \land \neg \text{Shr}[i] \land \neg \text{Exg} \land \text{Cmd} = \epsilon
\]

**Unsafe states:**
\[
\exists i, j. i \neq j \land \text{Cache}[i] = E \land \text{Cache}[j] \neq I
\]
*(cubes)*
Example: German-ish cache coherence protocol

Client $i$:
$\text{Cache}[i] \in \{E, S, I\}$

Directory:
$\text{Cmd} \in \{rs, re, \epsilon\}$
$\text{Ptr} \in \text{proc}$
$\text{Shr}[i] \in \{\text{true}, \text{false}\}$
$\text{Exg} \in \{\text{true}, \text{false}\}$

$t_5 : \exists i. \quad \text{Ptr} = i \land \text{Cmd} = rs \land \neg \text{Exg} \land \text{Cmd}' = \epsilon \land \text{Shr}'[i] \land \text{Cache}'[i] = S$
BRAB algorithm

I : initial states  \( U \) : unsafe states (cubes)  \( \mathcal{T} \) : transitions

\[ \text{BRAB}() : \]
\[
B := \emptyset; \quad \text{Kind}(U) := \text{Orig}; \quad \text{From}(U) := U;
\]
\[
\mathcal{M} := \text{FWD}(d_{max}, k);
\]
\[ \text{while BWDA()} = \text{unsafe} \text{ do} \]
\[
\quad \text{if Kind}(F) = \text{Orig} \text{ then return unsafe}
\]
\[
\quad B := B \cup \{ \text{From}(F) \}
\]
\[ \text{return safe} \]
BRAB algorithm

I : initial states    U : unsafe states (cubes)    $\mathcal{T}$ : transitions

BWD ():

$V := \emptyset$;

push(Q, U);

while not empty(Q) do

$\varphi := \text{pop}(Q)$;

if $\varphi \land I$ sat then return unsafe

if $\neg(\varphi \models \bigvee_{\psi \in V} \psi)$ then

$V := V \cup \{\varphi\}$;

push(Q, $\text{pre}_{\mathcal{T}}(\varphi)$);

return safe
BRAB algorithm

I : initial states   U : unsafe states (cubes)   T : transitions

BWDA ():

V := ∅;
push(Q, U);

while not empty(Q) do

φ := pop(Q);
if φ ∧ I sat then return unsafe

if ¬(φ ├ V ψ∈V ψ) then

V := V ∪ {φ};
push(Q, Approxₜ(φ));

return safe
BRAB algorithm

\[ I : \text{initial states} \quad U : \text{unsafe states (cubes)} \quad \mathcal{T} : \text{transitions} \]

\[ \text{Approx}_{\mathcal{T}} (\varphi) : \]

\[ \text{foreach } \psi \text{ in } \text{candidates}(\varphi) \text{ do} \]
\[ \text{if } \psi \notin B \land \mathcal{M} \not\models \psi \text{ then} \]
\[ \text{Kind}(\psi) := \text{Appr} ; \]
\[ \ldots \]
\[ \text{return } \psi \]
\[ \ldots \]
\[ \text{return } \pre_{\mathcal{T}} (\varphi) \]
Example: BRAB on German-*ish*
Example: BRAB on German-\textit{ish}

$\neg \text{Exg}
\quad \text{Cmd} = \epsilon
\quad \text{Cache}[\#1] = I
\quad \text{Cache}[\#2] = I
\quad \neg \text{Shr}[\#1]
\quad \neg \text{Shr}[\#2]$

$\exists i \neq j. \text{Cache}[i] = E$
$\text{Cache}[j] \neq I$
Example: BRAB on German-ish

∃i ≠ j. Cache[i] = E
Cache[j] ≠ 1
Example: BRAB on German-lish
Example: BRAB on German-ish
Example: BRAB on German-lish

\[ \neg \text{ExgCmd} = \epsilon \]
\[ \text{Cache}[\#1] = I \]
\[ \text{Cache}[\#2] = I \]
\[ \neg \text{Shr}[\#1] \]
\[ \neg \text{Shr}[\#2] \]

\[ \text{ExgCmd} = \epsilon \]
\[ \text{Ptr} = \#2 \]
\[ \text{Cache}[\#1] = I \]
\[ \text{Cache}[\#2] = I \]
\[ \neg \text{Shr}[\#1] \]
\[ \neg \text{Shr}[\#2] \]

\[ \text{ExgCmd} = \text{rs} \]
\[ \text{Ptr} = \#1 \]
\[ \text{Cache}[\#1] = I \]
\[ \text{Cache}[\#2] = I \]
\[ \neg \text{Shr}[\#1] \]
\[ \neg \text{Shr}[\#2] \]

\[ \exists i \neq j. \text{Cache}[i] = E \]
\[ \text{Cache}[j] \neq I \]
Example: BRAB on German-lish
Example: BRAB on German-lish

\[ \exists i. \text{Cmd} = \text{rs} \]
\[ \text{Cache}[i] = E \]

Extracting a candidate (\text{Approx}_\tau)
Example: BRAB on German-

ish

Exg
Cmd = rs
Ptr = #2
Cache[1] = E
Cache[2] = 1
Shr[1]
¬Shr[2]

Checking candidate

∃i. Cmd = rs
Cache[i] = E

∃i. Cmd = rs
Cache[i] = E

pre(t4(j))
pre(t5(j))
pre(t6(i))

∃i ≠ j. Cache[i] = E
Cache[j] ≠ 1
Example: BRAB on German-\textit{ish}
Example: BRAB on German-ish
Example: BRAB on German-lish
Example: BRAB on German-**ish**
Example: BRAB on German-lish
Example: BRAB on German-\textit{ish}
Example: BRAB on German-"ish"
Example: BRAB on German-lish
Example: BRAB on German-lish
Example: BRAB on German-lish
Example: BRAB on German-\textit{ish}
Example: BRAB on German-lish
Example: BRAB on German-**ish**
### Some benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>BRAB</th>
<th>Cubicle</th>
<th>CMurphi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Szymanski_at</td>
<td>0.14s</td>
<td>0.30s</td>
<td>8.04s (8) 5m12s (10) 2h50m (12)</td>
</tr>
<tr>
<td>German_Baukus</td>
<td>0.25s</td>
<td>7.03s</td>
<td>0.74s (4) 19m35s (8) 4h49m (10)</td>
</tr>
<tr>
<td>German.CTC</td>
<td>0.29s</td>
<td>3m23s</td>
<td>1.83s (4) 43m46s (8) 12h35m (10)</td>
</tr>
<tr>
<td>German_pfs</td>
<td>0.34s</td>
<td>3m58s</td>
<td>0.99s (4) 22m56s (8) 5h30m (10)</td>
</tr>
<tr>
<td>Chandra-Toueg</td>
<td>2m17s</td>
<td>2h01m</td>
<td>5.68s (4) 2m58s (5) 1h36m (6)</td>
</tr>
<tr>
<td>Szymanski_na</td>
<td>0.19s</td>
<td>T.O.</td>
<td>0.88s (4) 8m25s (6) 7h08m (8)</td>
</tr>
<tr>
<td>Flash_nodata</td>
<td>0.36s</td>
<td>O.M.</td>
<td>4.86s (3) 3m33s (4) 2h46m (5)</td>
</tr>
<tr>
<td>Flash</td>
<td>5m40s</td>
<td>O.M.</td>
<td>1m27s (3) 2h15m (4) O.M. (5)</td>
</tr>
</tbody>
</table>

O.M. > 20 GB
T.O. > 20 h
BRAB is complete only if the framework admits a complete Backward Reachability.

Cubicle goes beyond decidable fragment of array-based systems.

FLASH is expressed outside of this fragment.

BRAB remains safe.
Future work

Improvements:

- Experiment with real size industrial protocols
- Improve backtracking
- Difficult to discover candidates for numerical invariants

Certification:

- Deductive program verification (with Why3 and Alt-Ergo) and code extraction
- **Goal**: obtain a certified and efficient model checker
Thank you

Visit our web site
http://cubicle.lri.fr