Efficient Verification of Periodic Programs Using Sequential Consistency and Snapshots

Sagar Chaki, Arie Gurfinkel, Nishant Sinha
October 24, 2014

FMCAD’14, Lausanne, Switzerland
**Efficient Verification of Periodic Programs Using Sequential Consistency and Snapshots**

**Chaki /Sagar**

**Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213**

---

**13. SUPPLEMENTARY NOTES**

The original document contains color images.

---

**14. ABSTRACT**

---

**15. SUBJECT TERMS**

---

**16. SECURITY CLASSIFICATION OF:**

- a. REPORT unclassified
- b. ABSTRACT unclassified
- c. THIS PAGE unclassified

**17. LIMITATION OF ABSTRACT SAR**

**18. NUMBER OF PAGES 23**

---

**Standard Form 298 (Rev. 8-98)**

Prescribed by ANSI Std Z39-18
Outline

- Context
  - Periodic Programs
  - Time-Bounded Verification

- Verification Condition Generation
  - Hierarchical Lamport Clocks
  - Snapshotting

- Experimental Results

- Related Work
**Periodic Embedded Real-Time Software**

Automotive System

Rate Monotonic Scheduling (RMS)

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine control</td>
<td>10ms</td>
</tr>
<tr>
<td>Airbag</td>
<td>40ms</td>
</tr>
<tr>
<td>Braking</td>
<td>40ms</td>
</tr>
<tr>
<td>Cruise Control</td>
<td>50ms</td>
</tr>
<tr>
<td>Collision Detection</td>
<td>50ms</td>
</tr>
<tr>
<td>Entertainment</td>
<td>80ms</td>
</tr>
</tbody>
</table>

*Domains: Avionics, Automotive*

*OS: OSEK, VxWorks, RTEMS*

*We call them periodic programs*
Time-Bounded Verification [FMCAD’11&’14, VMCAI’13]

Input: Periodic Program
• Collection of periodic tasks
  • Execute concurrently with preemptive priority-based scheduling
  • Priorities respect RMS
  • Communicate through shared memory

Problem: Time-Bounded Verification
• Assertion A violated within X ms of a system’s execution from initial state I?
  • A, X, I are user specified
  • Time bounds map naturally to program’s functionality (e.g., air bags)

Solution: Bounded Model Checking
• Generate Verification Condition (SMT Formula over Bit-Vectors)
• Use SMT Solver to check satisfiability

Main focus of this paper
Periodic Program (PP)

An N-task periodic program PP is a set of tasks \( \{\tau_1, \ldots, \tau_N\}\)

A task \( \tau \) is a tuple \( \langle I, T, P, C, A \rangle \), where

- \( I \) is a task identifier = its priority
- \( T \) is a task body (i.e., code)
- \( P \) is a period
- \( C \) is the worst-case execution time
- \( A \) is the release time: the time at which task becomes first enabled

Semantics of PP bounded by time \( X \equiv \) asynchronous concurrent program:

\[
\begin{align*}
  & k_i = 0; \\
  & \text{while } (k_i < J_i \&\& \text{Wait}(\tau_i, k_i)) \\
  & \quad T_i(); \\
  & \quad k_i = k_i + 1;
\end{align*}
\]

blocks \( \tau_i \) until time \( A_i + k_i \times P_i \)

\[
J_i = \frac{X}{P_i}
\]
Periodic Program Example

\( \tau_1 = \langle 1, J_1, 8, 2, 0 \rangle, \quad \tau_2 = \langle 2, J_2 = J_3, 4, 1, 1 \rangle \)

Job1 of \( \tau_2 \)

Low-Priority Task

High-Priority Task

\( \tau_1 \)

\( \tau_2 \)

Legal Execution – \( \tau_1 \) executes for 2 units

Another Legal Execution – \( \tau_1 \) executes for 1 units

Illegal Execution – \( \tau_1 \) preempts \( \tau_2 \)
Verification Condition

\[ VC = VC_{seq} \land VC_{clk} \land VC_{obs} \]

- **Encodes Purely Job-local computation.** Value Read/Written by each Shared Variable access represented by a fresh variable.

- **Associates each shared variable access with a hierarchical Lamport Clock.** Constraints values of Clock components based on timing and priority.

- **Connects value read at each “Read” to the value written by most recent write according to the Lamport Clock.**
Verification Condition $VC_{seq}$

Same as verification condition for sequential program except that both reads and writes are given fresh variables

$J_1() \{ x := x + 1; \} \rightarrow x_2 = x_1 + 1$

$J_2() \{ x := x + 1; \} \rightarrow x_4 = x_3 + 1$

$J_3() \{ x := x + 1; \} \rightarrow x_6 = x_5 + 1$
Verification Condition $VC_{clk}$

\[ \tau_1 \]

\[ \begin{array}{c}
J_1 \\
\end{array} \]

\[ \tau_2 \]

\[ \begin{array}{cccccccc}
J_2 & & & & & & J_3 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array} \]

\[ \begin{array}{cccccccc}
x_1 & x_3 & x_4 & x_2 & & & x_5 & x_6 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array} \]

Observe: $x_i$ is accessed before $x_j$ iff 
\[
(R_i, \pi_i, \iota_i) < (R_j, \pi_j, \iota_j)
\]
where $<$ is lexicographic ordering

Claim/Intuition: This holds for all legal executions, not just this one.

- $\pi_i =$ priority of job accessing $x_i$
  - $\pi_1 = \pi_2 = 1, \pi_3 = \cdots = \pi_6 = 2$
- $R_i =$ # of jobs finished before $x_i$ accessed
  - $R_1 = R_3 = R_4 = 0, R_2 = 1, R_5 = R_6 = 2$
- $\iota_i =$ index of instruction accessing $x_i$ in topological ordering of CFG
  - $\iota_1 = \iota_3 = \iota_5 = 1, \iota_2 = \iota_4 = \iota_6 = 2$
Verification Condition $VC_{obs}$

Let $J_i$ = job in which $x_i$ is accessed

Compute: $J \sqsubseteq J'$ if $J$ always completes before $J'$ starts

Let $\kappa_i = (R_i, \pi_i, \iota_i)$ and for each read $x_i$, let

$W_i = \{x_j | x_j$ is a write $\land \neg(J_i \sqsubseteq J_j)\}$, i.e., the set of all writes that $x_i$ “may observe”

$\quad \quad VC_{obs} \equiv$

The value of each $x_i$ accessed by a read equals the value of $x_j$ such that $\kappa_j = \max\{\kappa_k | \kappa_k < \kappa_i$ and $x_k \in W_i\}$, where $\max\{} = initial$ value of $x$. 
Verification Condition $V C_{obs}$

For each read $x_i$ introduce $\tilde{\kappa}_i = \text{clock of write action observed}$

$$V C_{obs} \equiv$$

$$\land_{x_j \in W_i} \kappa_j < \kappa_i \Rightarrow \kappa_j \leq \tilde{\kappa}_i$$

$$\land$$

$$((V C^1_{obs}) \lor (\lor_{x_j \in W_i} V C^2_{obs}(j)))$$

\[
\begin{align*}
V C^1_{obs} & \equiv (\land_{x_j \in W_i} \kappa_j \geq \kappa_i) \land (x_i = x_{\text{Init}}) \\
V C^2_{obs}(j) & \equiv (\kappa_j < \kappa_i \land \kappa_j = \tilde{\kappa}_i) \land x_i = x_j
\end{align*}
\]

In the paper, we handle multiple shared variables.
Handling Locks

We handle two types of locks (both involve changing priorities)

- Each thread has a base priority = priority of task it executes
- Each PCP lock $l$ is associated with priority $\pi(l)$
  - A CPU lock is a PCP lock such that $\pi(l) = \infty$
  - Thread’s priority = max (its base priority, priorities of all PCP locks it holds)

Lock operation encoded by “priority-test-and-set” action $(J, pc, \pi_t, L_r, L_a)$

- Guard: All held locks must have priority less than $\pi_t$
- Command: Locks in $L_r$ are released; Locks in $L_a$ are acquired
- Encode by updating $VC_{clk}$ and $VC_{obs}$ appropriately

Note: To handle locks, we generalize VC-Gen to support operations that read and write program state (in this case held locks) atomically

- This will be useful for snapshotting (coming up)
Snapshotting: Problem

Sequence of jobs. Each job writes to a variable multiple times.

\[ J_1() \{ t := x; if(t) x := t + 1; else x := t + 2; \} \]
\[ \rightarrow \{ t := x_1; if(t) x_2 := t + 1; else x_3 := t + 2; \} \]

\[ J_2() \{ t := x; if(t) x := t + 1; else x := t + 2; \} \]
\[ \rightarrow \{ t := x_4; if(t) x_5 := t + 1; else x_6 := t + 2; \} \]

\[ J_n() \{ t := x; if(t) x := t + 1; else x := t + 2; \} \]
\[ \rightarrow \{ t := x_{3n-2}; if(t) x_{3n-1} := t + 1; else x_{3n} := t + 2; \} \]

Observe: \( W_1 = \{x_2, x_3\}, W_4 = \{x_2, x_3, x_5, x_6\}, W_7 = \{x_2, x_3, x_5, x_6, x_8, x_9\}, \ldots \)

Result: \( V_{C_{obs}} \) has large disjunctions with many redundant sub-formulas

Empirically: SMT solvers do not scale beyond small number of jobs
Snapshotting: Solution

Atomically read and write variable at the end of the job. Dominates all other access in the job.

\[ J_1() \{ t := x; \text{if} (t) x := t + 1; \text{else} x := t + 2; \text{atomic}: x := x; \} \]

\[ J_2() \{ t := x; \text{if} (t) x := t + 1; \text{else} x := t + 2; \} \]

\[ J_n() \{ t := x; \text{if} (t) x := t + 1; \text{else} x := t + 2; \} \]

Now: \( W_1 = W_4 = \{x_2, x_3\}, W_5 = W_8 = \{x_4, x_6, x_7\}, W_9 = W_{12} = \{x_8, x_{10}, x_{11}\}, \ldots \)

Result: \( VC_{obs} \) has smaller disjunctions with fewer redundant sub-formulas

Empirically: SMT solvers scale beyond small number of jobs

Choice of variables to snapshot: (i) all variables (ii) only written by the job
Verification Condition $VC_{obs}$ with Snapshotting

Input: $Snaps(J) =$ set of variables snapshotted by $J$
Compute: Relation $J \uparrow J'$ iff $J$ can be preempted by $J'$
Let $\Psi_{\subseteq}(J, g) =$ maximal jobs less that $J$ that snapshot $g$

Let $\Psi_{\uparrow}(J, g) = \{J' | J \uparrow J' \land g \in Snaps(J')\}$
Let $\Psi_{\downarrow}(J) = \{J' | J' = J \lor J' \uparrow J\}$

$W_i = \{x_j | x_j \text{ is a snapshot} \land J_j \in \Psi_{\uparrow}(J, g)\} \cup \{x_j | x_j \text{ is a snapshot} \land J_j \in \Psi_{\subseteq}(J, g)\} \cup \{x_j | x_j \text{ is a write} \land J_j \in \Psi_{\downarrow}(J, g)\}$

$VC_{obs} \equiv$ same as before with the new definition of $W_i$ above
## Results (Time in seconds)

<table>
<thead>
<tr>
<th></th>
<th>NONE</th>
<th>ALL</th>
<th>MOD</th>
<th>REKH</th>
</tr>
</thead>
<tbody>
<tr>
<td>nxt.bug1:H1</td>
<td>33</td>
<td>9</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>nxt.bug2:H1</td>
<td>32</td>
<td>10</td>
<td>7</td>
<td>31</td>
</tr>
<tr>
<td>nxt.ok1:H1</td>
<td>19</td>
<td>7</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>nxt.ok2:H1</td>
<td>20</td>
<td>7</td>
<td>6</td>
<td>29</td>
</tr>
<tr>
<td>nxt.ok3:H1</td>
<td>30</td>
<td>8</td>
<td>6</td>
<td>31</td>
</tr>
<tr>
<td>aso.bug1:H1</td>
<td>29</td>
<td>9</td>
<td>9</td>
<td>34</td>
</tr>
<tr>
<td>aso.bug2:H1</td>
<td>28</td>
<td>10</td>
<td>9</td>
<td>32</td>
</tr>
<tr>
<td>aso.bug3:H1</td>
<td>29</td>
<td>13</td>
<td>11</td>
<td>80</td>
</tr>
<tr>
<td>aso.bug4:H1</td>
<td>32</td>
<td>17</td>
<td>9</td>
<td>66</td>
</tr>
<tr>
<td>aso.ok1:H1</td>
<td>32</td>
<td>11</td>
<td>10</td>
<td>32</td>
</tr>
<tr>
<td>aso.ok2:H1</td>
<td>38</td>
<td>29</td>
<td>17</td>
<td>67</td>
</tr>
<tr>
<td>nxt.bug1:H4</td>
<td>*</td>
<td>119</td>
<td>74</td>
<td>*</td>
</tr>
<tr>
<td>nxt.bug2:H4</td>
<td>*</td>
<td>172</td>
<td>92</td>
<td>*</td>
</tr>
<tr>
<td>nxt.ok1:H4</td>
<td>*</td>
<td>89</td>
<td>49</td>
<td>*</td>
</tr>
</tbody>
</table>

NONE=No snapshotting, ALL=Snapshot all variables, MOD=Snapshot only modified variables, REKH=Previous tool based on sequentialization

2GB Memory Limit
60min Time Limit
### Results (Time in seconds)

<table>
<thead>
<tr>
<th>Snapshot</th>
<th>NONE</th>
<th>ALL</th>
<th>MOD</th>
<th>REKH</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntx.ok2:H4</td>
<td>* 125</td>
<td>49</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>ntx.ok3:H4</td>
<td>* 358</td>
<td>133</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>aso.bug1:H4</td>
<td>* 128</td>
<td>92</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>aso.bug2:H4</td>
<td>* 147</td>
<td>74</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>aso.bug3:H4</td>
<td>* 209</td>
<td>136</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>aso.bug4:H4</td>
<td>* 329</td>
<td>152</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>aso.ok1:H4</td>
<td>* 270</td>
<td>210</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>aso.ok2:H4</td>
<td>*</td>
<td>* 1312</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>ctm.bug2</td>
<td>36 29</td>
<td>21</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>ctm.bug3</td>
<td>* 124</td>
<td>59</td>
<td>258</td>
<td></td>
</tr>
<tr>
<td>ctm.ok1</td>
<td>23</td>
<td>37</td>
<td>21</td>
<td>122</td>
</tr>
<tr>
<td>ctm.ok2</td>
<td>28</td>
<td>26</td>
<td>17</td>
<td>111</td>
</tr>
<tr>
<td>ctm.ok3</td>
<td>* 116</td>
<td>53</td>
<td>275</td>
<td></td>
</tr>
<tr>
<td>ctm.ok4</td>
<td>* 320</td>
<td>143</td>
<td>395</td>
<td></td>
</tr>
</tbody>
</table>

**NONE**=No snapshotting, **ALL**=Snapshot all variables, **MOD**=Snapshot only modified variables, **REKH**=Previous tool based on sequentialization

- **2GB Memory Limit**
- **60min Time Limit**
# Observability Sizes

|                  | \(\text{AVG OBS}(\mathcal{P})\) | \(|W(\mathcal{P})|\) |
|------------------|---------------------------------|-----------------------|
| \(\text{nxt.bug1:H1}\) | \begin{tabular}{c|c|c} \text{NONE} & \text{ALL} & \text{MOD} \end{tabular} | \begin{tabular}{c|c|c} \text{NONE} & \text{ALL} & \text{MOD} \end{tabular} |
|                  | \text{25.6} & \text{2.9} & \text{2.9} | \text{298} & \text{455} & \text{416} |
| \(\text{nxt.bug2:H1}\) | \text{26.5} & \text{3.1} & \text{3.2} | \text{310} & \text{492} & \text{429} |
| \(\text{nxt.ok1:H1}\) | \text{25.6} & \text{2.9} & \text{2.9} | \text{298} & \text{455} & \text{416} |
| \(\text{nxt.ok2:H1}\) | \text{25.4} & \text{3.0} & \text{2.9} | \text{298} & \text{454} & \text{416} |
| \(\text{nxt.ok3:H1}\) | \text{26.5} & \text{3.1} & \text{3.2} | \text{310} & \text{492} & \text{429} |
| \(\text{aso.bug1:H1}\) | \text{26.0} & \text{3.6} & \text{3.6} | \text{304} & \text{512} & \text{427} |
| \(\text{aso.bug2:H1}\) | \text{26.4} & \text{3.7} & \text{3.7} | \text{308} & \text{516} & \text{431} |
| \(\text{aso.bug3:H1}\) | \text{25.5} & \text{3.6} & \text{3.5} | \text{355} & \text{615} & \text{504} |
| \(\text{aso.bug4:H1}\) | \text{26.5} & \text{4.6} & \text{4.4} | \text{309} & \text{543} & \text{434} |
| \(\text{aso.ok1:H1}\) | \text{27.1} & \text{4.1} & \text{4.2} | \text{311} & \text{519} & \text{434} |
| \(\text{aso.ok2:H1}\) | \text{26.5} & \text{4.6} & \text{4.4} | \text{311} & \text{545} & \text{436} |
| \(\text{nxt.bug1:H4}\) | \text{99.5} & \text{3.0} & \text{3.0} | \text{1192} & \text{1835} & \text{1676} |
| \(\text{nxt.bug2:H4}\) | \text{102.9} & \text{3.1} & \text{3.2} | \text{1240} & \text{1989} & \text{1731} |
| \(\text{nxt.ok1:H4}\) | \text{99.5} & \text{3.0} & \text{3.0} | \text{1192} & \text{1835} & \text{1676} |

\[\text{AVGOBS}(\mathcal{P}) = \text{avg. no. of reads observing each write or snapshot}\]

\[|W(\mathcal{P})| = \text{total no. of snapshot and write variables}\]
## Observability Sizes

|                | $\text{AVGOBS}(\mathcal{P})$ | $|\mathcal{W}(\mathcal{P})|$ |
|----------------|-------------------------------|-------------------------------|
|                | NONE | ALL | MOD | NONE | ALL | MOD |
| nxt.ok2:H4     | 99.3 | 3.0 | 3.0 | 1192 | 1834 | 1675 |
| nxt.ok3:H4     | 102.9 | 3.1 | 3.2 | 1240 | 1989 | 1731 |
| aso.bug1:H4    | 99.9 | 3.6 | 3.6 | 1216 | 2072 | 1723 |
| aso.bug2:H4    | 101.6 | 3.7 | 3.7 | 1232 | 2088 | 1739 |
| aso.bug3:H4    | 98.3 | 3.6 | 3.5 | 1420 | 2490 | 2034 |
| aso.bug4:H4    | 100.4 | 4.6 | 4.4 | 1236 | 2199 | 1751 |
| aso.ok1:H4     | 103.2 | 4.1 | 4.2 | 1244 | 2100 | 1751 |
| aso.ok2:H4     | 100.1 | 4.6 | 4.4 | 1244 | 2207 | 1759 |
| ctm.bug2       | 17.9  | 4.1 | 4.5 | 512  | 1052 | 683  |
| ctm.bug3       | 26.6  | 4.1 | 4.5 | 768  | 1588 | 1033 |
| ctm.ok1        | 18.6  | 4.1 | 4.6 | 512  | 1052 | 684  |
| ctm.ok2        | 18.1  | 4.1 | 4.5 | 512  | 1052 | 683  |
| ctm.ok3        | 27.9  | 4.1 | 4.5 | 780  | 1600 | 1057 |
| ctm.ok4        | 36.4  | 4.2 | 4.7 | 1040 | 2140 | 1400 |

$\text{AVGOBS}(P) =$ avg. no. of reads observing each write or snapshot

$|\mathcal{W}(P)| =$ total no. of snapshot and write variables
Related Work

Generate Verification Condition by Encoding Dataflow between Reads and Writes Using Lamport Clocks
- Nishant Sinha, Chao Wang: Staged concurrent program analysis. SIGSOFT FSE 2010: 47-56

Generate Verification Condition per Scheduling round using prophecy variables, and ensure that output of one round equals input to the next

- Snapshotting combines both ideas
- Interplay between Logical Clocks and Prophecy Variables
  - Both due to Lamport
QUESTIONS?
Contact Information Slide Format

Sagar Chaki
Principal Researcher
SSD/CSC
Telephone: +1 412-268-1436
Email: chaki@sei.cmu.edu

Web
www.sei.cmu.edu
www.sei.cmu.edu/contact.cfm

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Customer Relations
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257