Efficient Verification of Periodic Programs Using Sequential Consistency and Snapshots

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

FMCAD'14, Lausanne, Switzerland

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **24 OCT 2014**	2. REPORT TYPE **N/A**	3. DATES COVERED
4. TITLE AND SUBTITLE **Efficient Verification of Periodic Programs Using Sequential Consistency and Snapshots**		5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Chaki /Sagar**		5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited.

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**SAR**	**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)

Task	Period
Engine control	10ms
Airbag	40ms
Braking	40ms
Cruise Control	50ms
Collision Detection	50ms
Entertainment	80ms

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 τ 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:

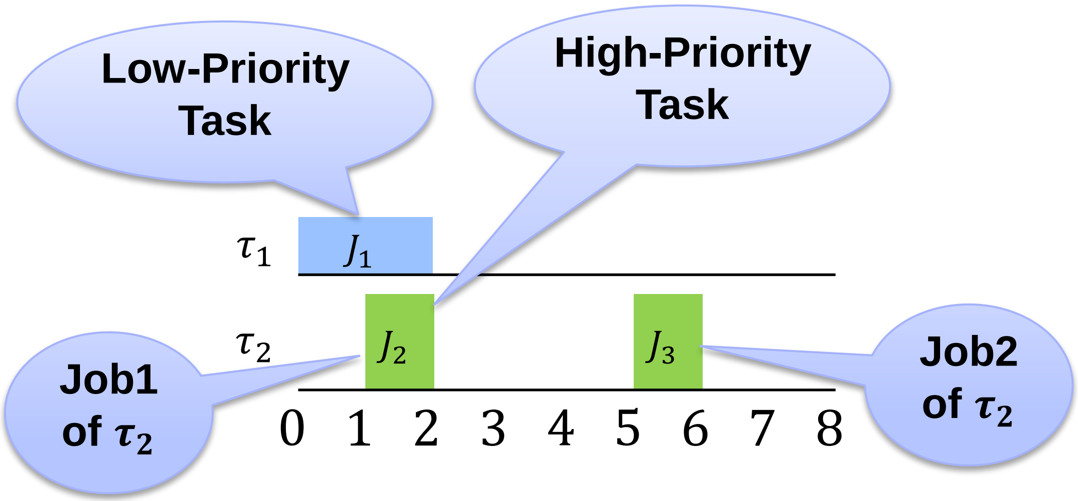
parallel execution w/ priorities

```
||  k_i = 0;
    while (k_i < J_i && Wait(τ_i, k_i))
      T_i ();
      k_i = k_i + 1;
```

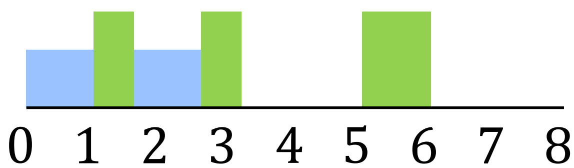
blocks τ_i until time $A_i + k_i \times P_i$

$$J_i = \frac{X}{P_i}$$

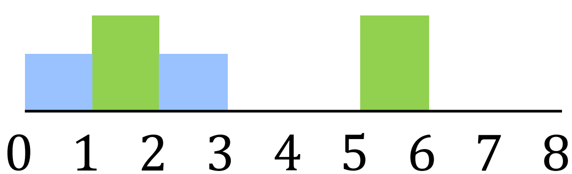
Periodic Program Example



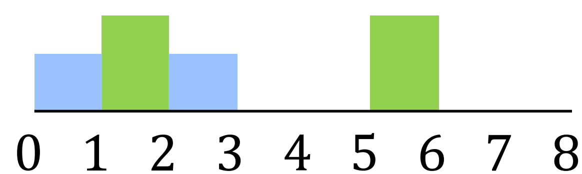
Illegal Execution – τ_1 preempts τ_2



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



Legal Execution – τ_1 executes for 2 units



Another Legal Execution – τ_1 executes for 1 units

Verification Condition

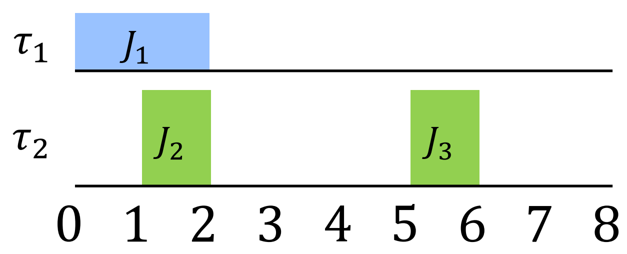
$$VC = VC_{seq} \wedge VC_{clk} \wedge VC_{obs}$$

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

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

$\wedge$

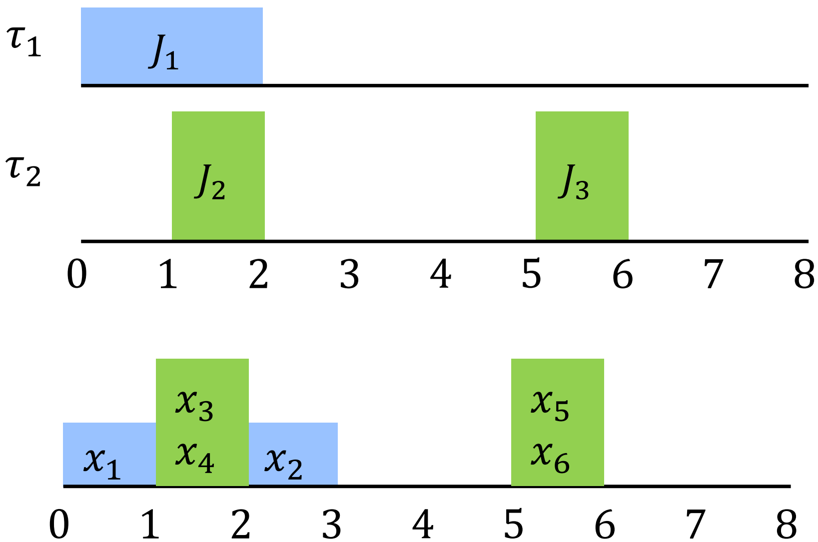
$J_2()\,\{\, x := x + 1; \}$ ⟶ $x_4 = x_3 + 1$

$\wedge$

$J_3()\,\{\, x := x + 1; \}$ ⟶ $x_6 = x_5 + 1$

(the three conjuncts together form VC_{seq})

Verification Condition VC_{clk}



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.

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

} VC_{clk}

Verification Condition VC_{obs}

Let J_i = job in which x_i is accessed

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

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

$W_i = \{x_j | x_j \; is \; a \; write \wedge \neg(J_i \sqsubset J_j)\}$, i.e., the set of all writes that x_i "may observe"

$$VC_{obs} \equiv$$

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 VC_{obs}

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

$$VC_{obs} \equiv$$
$$\bigwedge_{x_j \in W_i} \kappa_j < \kappa_i \Rightarrow \kappa_j \leq \tilde{\kappa}_i$$
$$\wedge$$
$$\left(\left(VC^1_{obs}\right) \vee \left(\bigvee_{x_j \in W_i} VC^2_{Obs}(j)\right)\right)$$

x_i observes initial value x_{Init} of x

$$VC^1_{obs} \equiv \left(\bigwedge_{x_j \in W_i} \kappa_j \geq \kappa_i\right) \wedge \left(x_i = x_{Init}\right)$$

$$VC^2_{Obs}(j) \equiv \left(\kappa_j < \kappa_i \wedge \kappa_j = \tilde{\kappa}_i\right) \wedge x_i = x_j$$

x_i observes x_j

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, π_t, L_r, L_a)

- Guard: All held locks must have priority less than π_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.

$J_1()\,\{\, t := x; if(t)\; x := t+1;\; else\;\; x := t+2;\, \}$ → $\{\, t := x_1; if(t)\; x_2 := t+1;\; else\;\; x_3 := t+2;\, \}$

$J_2()\,\{\, t := x; if(t)\; x := t+1;\; else\;\; x := t+2;\, \}$ → $\{\, t := x_4; if(t)\; x_5 := t+1;\; else\;\; x_6 := t+2;\, \}$

⋮

$J_n()\,\{\, t := x; if(t)\; x := t+1;\; else\;\; x := t+2;\, \}$ → $\{\, t := x_{3n-2}; if(t)\; x_{3n-1} := t+1;\; else\;\; x_{3n} := t+2;\, \}$

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: VC_{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.

$J_1()\,\{\, t := x; if(t)\ x := t + 1;$
$\quad else\ \ x := t + 2;$
$\quad atomic\!: x := x;\ \}$ → $\{\, t := x_1; if(t)\ x_2 := t + 1;$
$\quad else\ \ x_3 := t + 2;$
$\quad x_4 := x_4; \}$

$J_2()\,\{\, t := x; if(t)\ x := t + 1;$
$\quad else\ \ x := t + 2;\ \}$ → $\{\, t := x_5; if(t)\ x_6 := t + 1;$
$\quad else\ \ x_7 := t + 2;$
$\quad x_8 := x_8; \}$

⋮

$J_n()\,\{\, t := x; if(t)\ x := t + 1;$
$\quad else\ \ x := t + 2;\ \}$ → $\{\, t := x_{4n-3}; if(t)\ x_{4n-2} := t + 1;$
$\quad else\ \ x_{4n-1} := t + 2;$
$\quad x_{4n} := x_{4n}; \}$

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 $\Psi_{\sqsubset}(J, g)$ = maximal jobs less that J that snapshot g

Let $\Psi_{\uparrow}(J, g) = \{J' \mid J \uparrow J' \wedge g \in Snaps(J')\}$

Let $\Psi_{\downarrow}(J) = \{J' \mid J' = J \vee J' \uparrow J\}$

$$W_i = \left\{x_j \,\middle|\, x_j \; is \; a \; snapshot \wedge J_j \in \Psi_{\uparrow}(J, g)\right\} \cup \left\{x_j \,\middle|\, x_j \; is \; a \; snapshot \wedge J_j \in \Psi_{\sqsubset}(J, g)\right\} \cup \left\{x_j \,\middle|\, x_j \; is \; a \; write \wedge J_j \in \Psi_{\downarrow}(J, g)\right\}$$

$VC_{obs} \equiv$ same as before with the new definition of W_i above

Results (Time in seconds)

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

2GB Memory Limit

60min Time Limit

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

Results (Time in seconds)

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

2GB Memory Limit

60min Time Limit

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

Observability Sizes

	AvgObs($\mathcal{P}$)			$\lvert W(\mathcal{P}) \rvert$		
	NONE	ALL	MOD	NONE	ALL	MOD
nxt.bug1:H1	25.6	2.9	2.9	298	455	416
nxt.bug2:H1	26.5	3.1	3.2	310	492	429
nxt.ok1:H1	25.6	2.9	2.9	298	455	416
nxt.ok2:H1	25.4	3.0	2.9	298	454	415
nxt.ok3:H1	26.5	3.1	3.2	310	492	429
aso.bug1:H1	26.0	3.6	3.6	304	512	427
aso.bug2:H1	26.4	3.7	3.7	308	516	431
aso.bug3:H1	25.5	3.6	3.5	355	615	504
aso.bug4:H1	26.5	4.6	4.4	309	543	434
aso.ok1:H1	27.1	4.1	4.2	311	519	434
aso.ok2:H1	26.5	4.6	4.4	311	545	436
nxt.bug1:H4	99.5	3.0	3.0	1192	1835	1676
nxt.bug2:H4	102.9	3.1	3.2	1240	1989	1731
nxt.ok1:H4	99.5	3.0	3.0	1192	1835	1676

$AVGOBS(P)$ = avg. no. of reads observing each write or snapshot
$|W(P)|$ = total no. of snapshot and write variables

Observability Sizes

	AVGOBS($\mathcal{P}$)			$\lvert W(\mathcal{P}) \rvert$		
	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

$AVGOBS(P)$ = avg. no. of reads observing each write or snapshot
$|W(P)|$ = 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

- Akash Lal, Thomas W. Reps: Reducing Concurrent Analysis Under a Context Bound to Sequential Analysis. CAV 2008: 37-51

- **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

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

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

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