Compositional Sequentialization of Periodic Programs

Sagar Chaki[1], Arie Gurfinkel[1],
Soonho Kong[2], Ofer Strichman[3]

Jan 22, 2013

[1]Software Engineering Institute, CMU
[2]Computer Science Department, CMU
[3]Technion, Israel Institute of Technology

Software Engineering Institute | Carnegie Mellon

Time-Bounded Verification of Periodic Programs

Periodic Program

- Collection of periodic tasks
  - Execute concurrently with fixed-priority scheduling
  - Priorities respect RMS
  - Communicate through shared memory
  - Synchronize through preemption and priority ceiling locks

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)

Assumptions

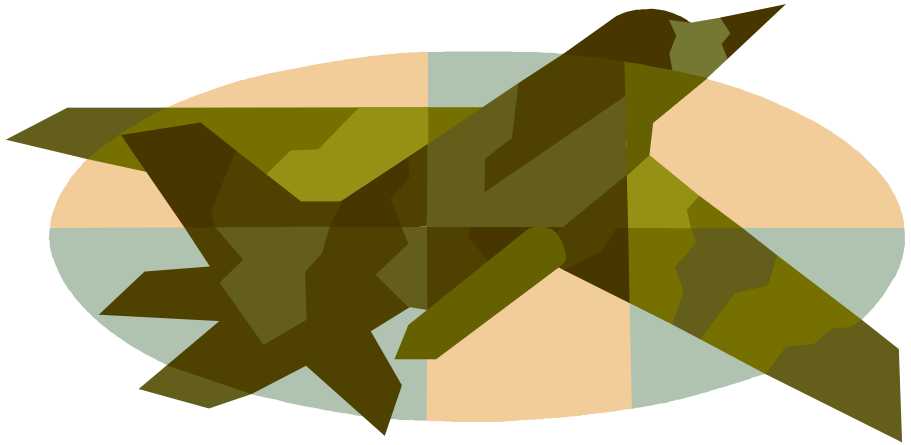
- System is schedulable
- WCET of each task is given

Motivation: Real-Time Embedded Systems

Avionics Mission System*

Rate Monotonic Scheduling (RMS)

Task	Period
weapon release	10ms
radar tracking	40ms
target tracking	40ms
aircraft flight data	50ms
display	50ms
steering	80ms



*Locke, Vogel, Lucas, and Goodenough. “Generic Avionics Software Specification”. SEI/CMU Technical Report CMU/SEI-90-TR-8-ESD-TR-90-209, December, 1990

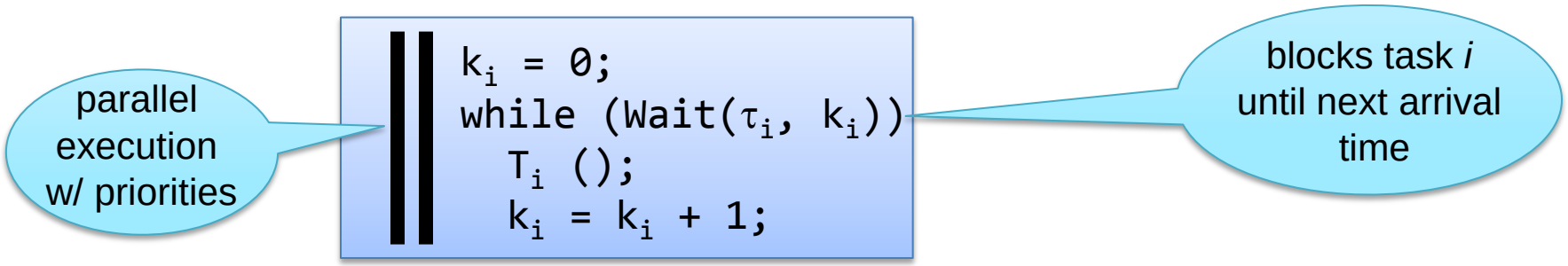
Periodic Program (PP)

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
- 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 is given by an asynchronous concurrent program:



Hyper-period = Least Common Multiple of all periods

- Program is *harmonic* if periods are multiples of each other

Time Bounded Semantics of Periodic Program

Assumptions

- (A1) Time window W is divisible by the hyper-period (i.e., W | H)
- (A2) Each task arrives in time to complete in 1st period (i.e., $A_i + RT_i \leq P_i$)

The time bound imposes a natural bound on # of jobs: $J_i = W / P_i$

Time-Bounded Semantics of PP is

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

Job-Bounded Abstraction

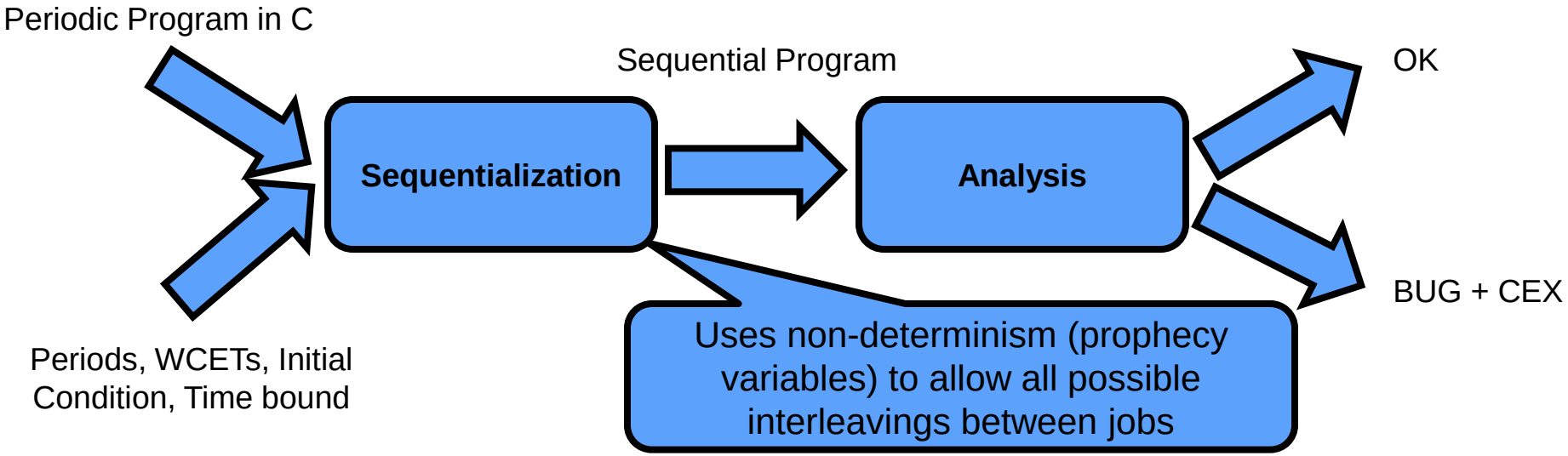
- Abstracts away time
- Approximates Wait() by a non-deterministic delay
- Preserves logical (time-independent) properties!

Our tool: REK

Supports C programs w/ tasks, priorities, priority ceiling protocol, shared variables

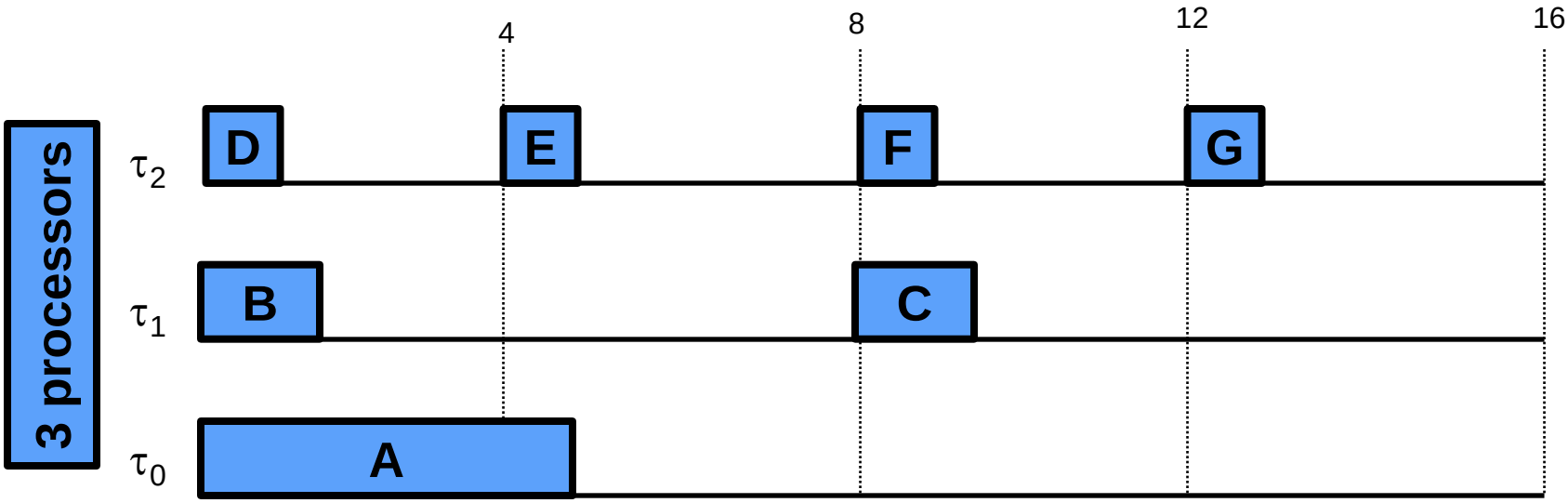
Works in two stages:

1. *Sequentialization* – reduction to sequential program w/ *prophecy* variables
2. *Bounded program analysis*: CBMC, HAVOC, others



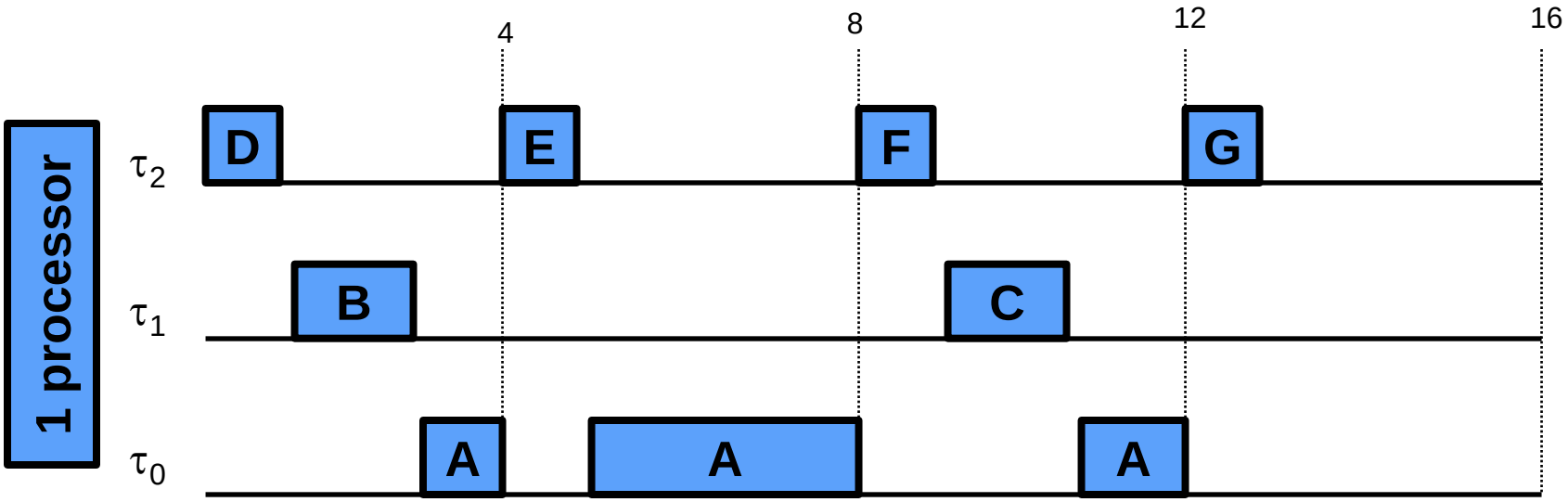
Contribution 1: Compositional Sequentialization – allows fewer interleavings between tasks and shorter counterexamples without losing soundness
Contribution 2: Empirical evaluation showing improvement

Example: A Harmonic PP



Task	WCET (C_i)	Period (P_i)	Arrival Time (A_i)
τ_2	1	4	0
τ_1	2	8	0
τ_0	5	16	0

Example: One Task Schedule



Task	WCET (C_i)	Period (P_i)	Arrival Time (A_i)
τ_2	1	4	0
τ_1	2	8	0
τ_0	5	16	0

Compositional sequentialization

Leverages two types of temporal separation between jobs

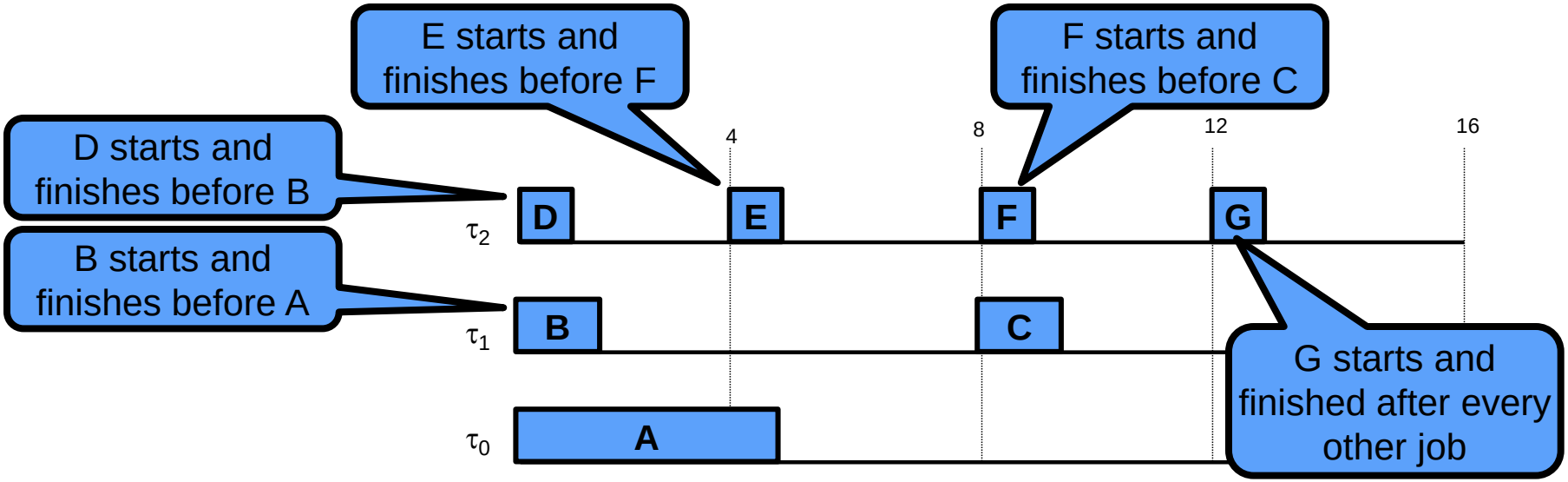
Intra-Hyper-Period

- Between jobs within the same hyper-period
- Prevents certain jobs in the same hyper-period from interleaving based on their priorities, arrival times, and worst-case execution times

Inter-Hyper-Period

- Between jobs across different hyper-periods
- Prevents interleaving between jobs from different hyper-periods
- Relies on assumption A2, which guarantees that all jobs in hyper-period i complete before any job in hyper-period (i+1) starts.

Intra-Hyper-Period Temporal Separation



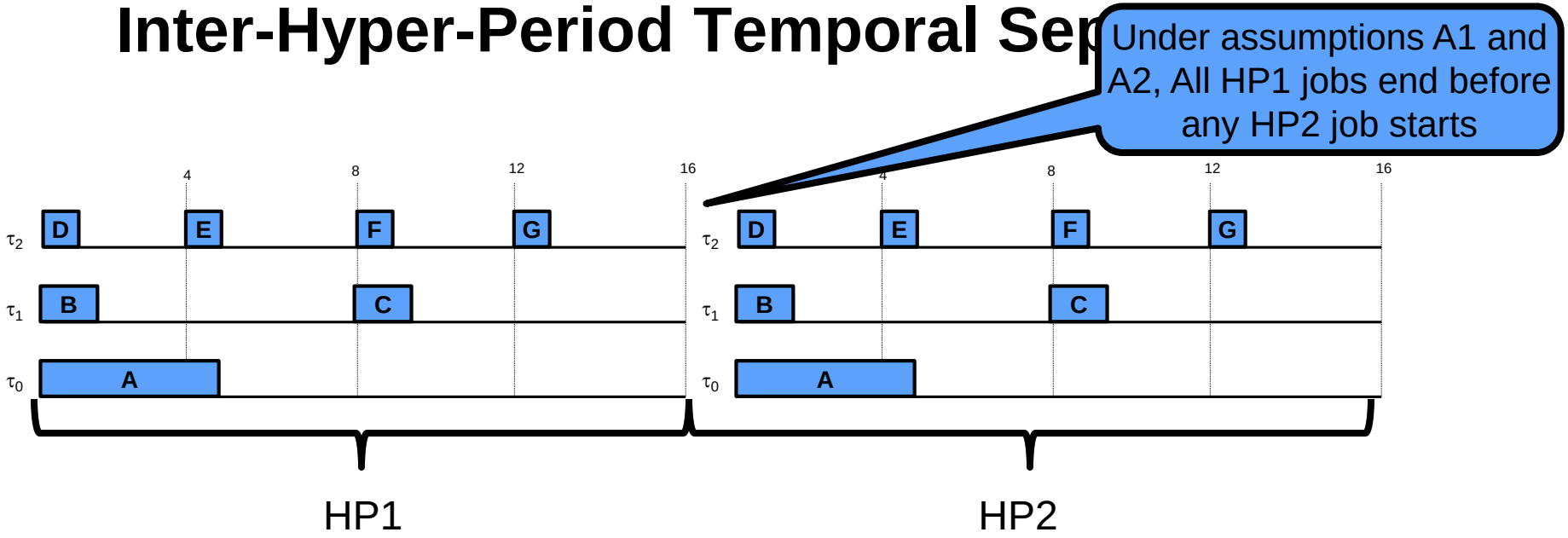
Monolithic Sequentialization (FMCAD11)

(A) || (B;C) || (D;E;F;G)

Compositional Sequentialization (VMCAI13)

D;B; (A || (E;F;C)); G

Inter-Hyper-Period Temporal Sep



Monolithic Sequentialization (FMCAD11)

(A;A) || (B;C;B;C) || (D;E;F;G; D;E;F;G)

Compositional Sequentialization (VMCAI13)

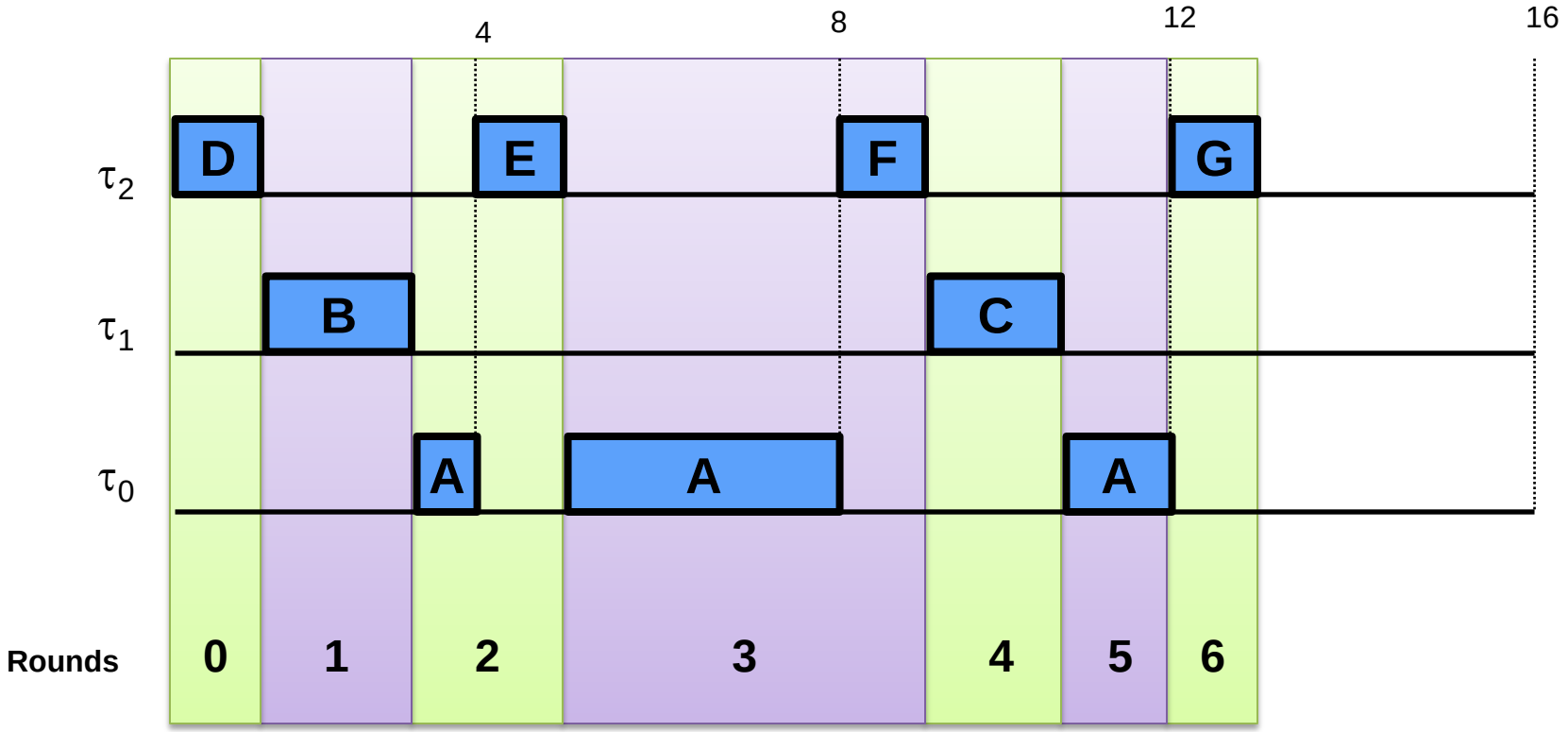
D;B; (A || (E;F;C)); G ; D;B; (A || (E;F;C)); G

Partition Execution into Rounds

Execution starts in round 0

A round ends, and a new one begins, each time a job finishes

- # rounds == # of jobs



Compositional Sequentialization

Define three job orderings based on priorities, WCET, and arrival time: $\lhd$, $\uparrow$, $\sqsubset$

Sequential Program for execution of R rounds:

1. for each global variable g, let g[i] be the value of g in round i

2. **(ScheduleJobs)** choose for each job j
   - start round: start[j]
   - end round: end[j]

Constrained by $\lhd$, $\uparrow$, $\sqsubset$

3. **(RunJobs)** execute job bodies sequentially
   - in some well-defined total order
   - for global variables, use g[i] instead of g when running in round i
   - non-deterministically decide where to context switch
   - at a context switch jump to a new round (cannot preempt a h

Ordered by $\sqsubset$

Done as soon as job ending at round i is over.

4. **(CheckAssumptions)** check that initial value of round i+1 is the final value of round i

5. **(CheckAssertions)** check user assertions

Done as soon as job (containing the assertion) and step 4 are over.

Job Ordering

Priority

Departure Time

$$J_1 \lhd J_2 \equiv (\pi(J_1) \leq \pi(J_2) \wedge D(J_1) \leq D(J_2)) \vee$$
$$(\pi(J_1) > \pi(J_2) \wedge A(J_1) \leq A(J_2))$$

Arrival Time

- J_1 completes before J_2 starts

$$J_1 \uparrow J_2 \equiv (\pi(J_1) < \pi(J_2) \wedge A(J_1) < A(J_2) < D(J_1))$$

- J_1 "could be" (due to WCET) preempted by J_2

$$J_1 \sqsubset J_2 \equiv (J_1 \lhd J_2) \vee (J_1 \uparrow J_2)$$

- Total order: lexicographic by (A,-π) (see Lemma 1)

Compositional Sequentialization: ScheduleJobs

1: **function** SCHEDULEJOBS( )

2: $\forall j \in \mathsf{J} \,.\, start[j] = *; end[j] = *$

// *Jobs are sequential*

3: $\forall i \in [0, N) \,.\, \forall k \in [0, J_i) \,.\, \text{assume}$
$(0 \cdot start[\mathsf{J}(i,k)] \cdot end[\mathsf{J}(i,k)] < R)$

// *Jobs are well-separated*

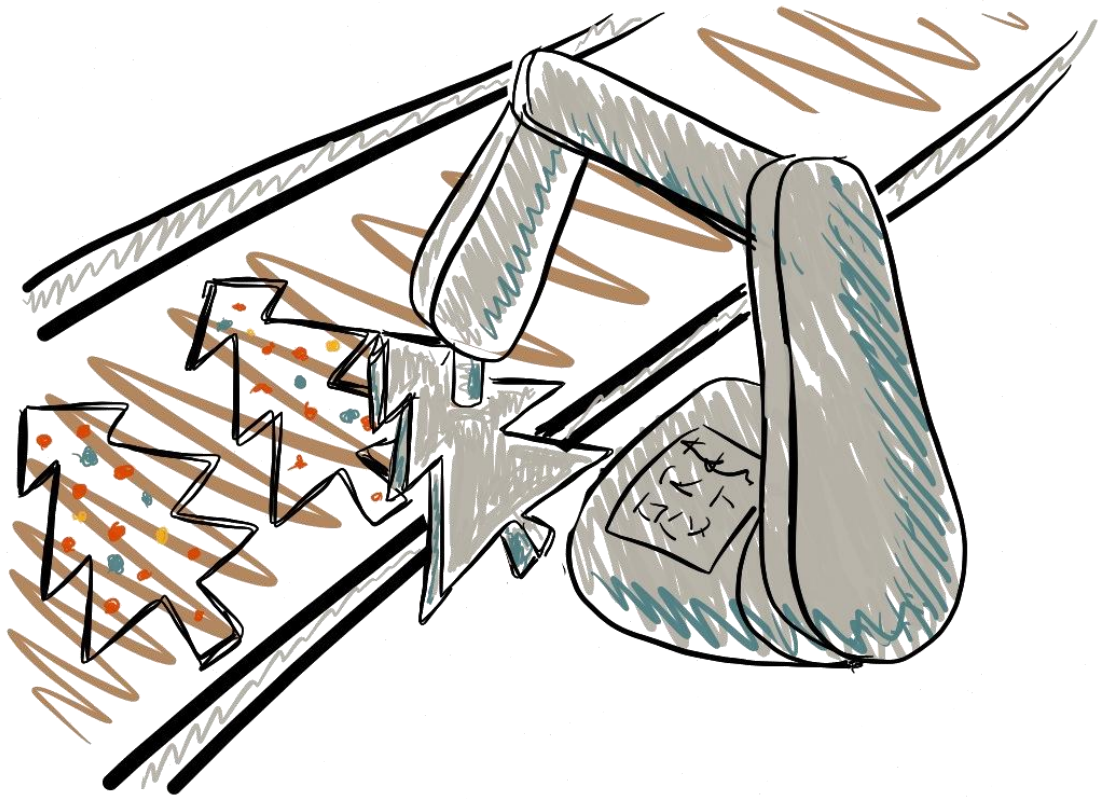
4: $\forall j_1 \lhd j_2 \,.\, \text{assume}(end[j_1] < start[j_2])$

5: $\forall j_1 \uparrow j_2 \,.\, \text{assume}(start[j_1] \cdot start[j_2])$

// *Jobs are well-nested*

6: $\forall j_1 \uparrow j_2 \,.\, \text{assume}(start[j_2] \cdot end[j_1]$
$\Longrightarrow (start[j_2] \cdot end[j_2] < end[j_1]))$

Case Study: A Metal Stamping Robot

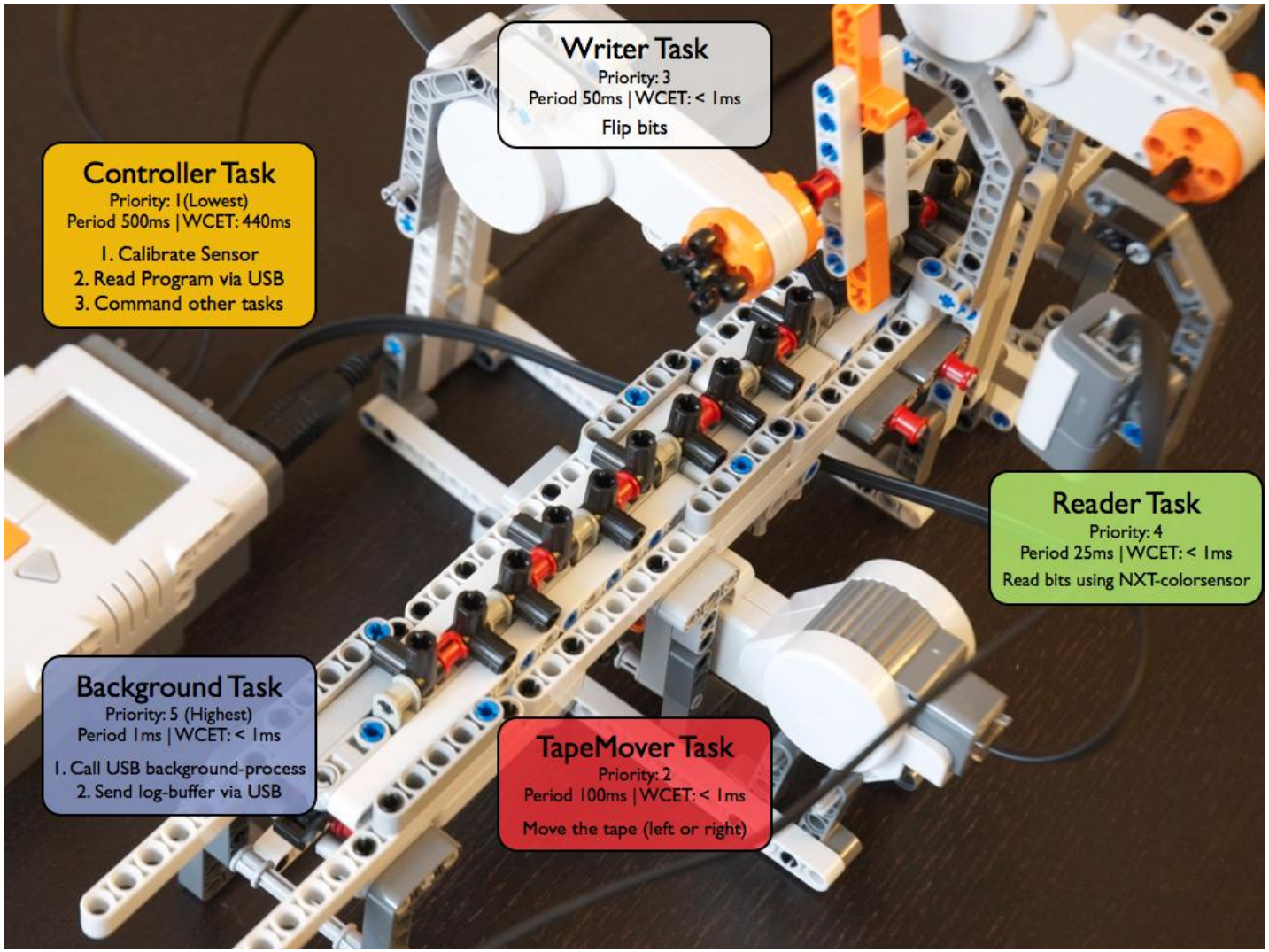


a.k.a. LEGO Turing Machine

http://www.youtube.com/watch?v=teDyd0d5M4o

Image courtesy of Taras Kowaliw

Turing Machine: Task Structure



An Example Property

When writer flips a bit, the tape motor and read motor should **stop**.

Controller Task

```
case C_WRITE:

  /* Check if we need to chagne the bit */
  if(R(input) != R(output)) {
    /* Check the header and move it back if necessary */
    if(nxt_motor_get_count(READ_MOTOR) > 0 && R(R_state) == READ_IDLE) {
      W(R_state, READ_MOVE_HEADER_BACKWARD);
    }

    /* Check the header and flip the bit if it is safe to do */
    if(nxt_motor_get_count(READ_MOTOR) <= 0 && R(W_state) == WRITE_IDLE) {
      W(W_state, WRITE_FLIP);
    }
  } else {
    /* Nothing to change for writer */
    W(W_state, WRITE_IDLE);
    C_state = C_MOVE;
  }
  break;
```

Writer Task

```
  case WRITE_FLIP:
#ifdef VERIFICATION
    /* Property 3: When writer flips a bit, the tape motor and read
       motor should be stopped. */

    /* FAILED!! with BOUND 120 */
    assert(R(T_speed) == 0 && R(R_speed) == 0);
#endif
```

NXTway-GS: a 2 wheeled self-balancing robot

Original: nxt (2 tasks)

- *balancer* (4ms)
  - Keeps the robot upright and responds to BT commands
- *obstacle* (50ms)
  - monitors sonar sensor for obstacle and communicates with *balancer* to back up the robot

Ours: aso (3 tasks)

- *balancer* as above but no BT
- *obstacle* as above
- *bluetooth* (100ms)
  - responds to BT commands and communicates with the *balancer*

Verified consistency of communication between tasks

Results: Turing Machine

			MONOSEQ						HARMONICSEQ					
Name				SAT Size			S	Time		SAT Size			S	Time
	H	OL	SL	GL	Var	Clause		(sec)	SL	GL	Var	Clause		(sec)
ctm.ok1	4	613	13K	121K	2,737K	8,774K	Y	44,781	7K	111K	1,063K	3,497K	Y	93.39
ctm.ok2	4	610	13K	119K	2,728K	8,738K	Y	21,804	7K	109K	1,055K	3,467K	Y	87.60
ctm.bug2	4	611	13K	118K	2,707K	8,674K	N	2,281	7K	108K	1,047K	3,441K	N	86.18
ctm.ok3	6	612	20K	222K	6,276K	20,163K	U	—	7K	171K	1,667K	5,566K	Y	243.76
ctm.bug3	6	612	20K	214K	5,914K	19,044K	N	84,625	7K	165K	1,609K	5,383K	N	248.65
ctm.ok4	8	613	29K	333K	10,390K	33,550K	U	—	7K	222K	2,178K	7,417K	Y	534.38

400x speedup

Timeout (1 day)

success

Results: Self-Balancing Robot

		MONOSEQ						HARMONICSEQ					
Name		SAT Size				S	Time	SAT Size				S	Time
	OL	SL	GL	Var	Clause		(sec)	SL	GL	Var	Clause		(sec)
1 hyper-period													
nxt.ok1	396	2,158	12K	128K	399K	Y	21.22	2,378	17K	110K	354K	Y	4.22
nxt.bug1	398	2,158	12K	128K	399K	N	6.22	2,378	17K	110K	354K	N	4.36
nxt.ok2	388	2,215	12K	132K	410K	Y	11.16	2,432	18K	111K	356K	Y	4.69
nxt.bug2	405	2,389	15K	135K	422K	N	8.66	2,704	23K	114K	372K	N	5.81
nxt.ok3	405	2,389	15K	135K	425K	Y	14.46	2,704	23K	109K	358K	Y	5.71
aso.bug1	421	2,557	17K	167K	541K	N	12.05	3,094	29K	173K	568K	N	6.67
aso.bug2	421	2,627	17K	167K	539K	N	11.61	3,184	29K	165K	549K	N	6.71
aso.ok1	418	2,561	17K	164K	525K	Y	22.20	3,098	28K	147K	486K	Y	6.51
aso.bug3	445	3,118	24K	350K	1,117K	N	22.15	4,131	41K	341K	1,108K	Y	19.27
aso.bug4	444	3,105	23K	325K	1,027K	N	16.32	4,118	40K	307K	1,001K	N	19.83
aso.ok2	443	3,106	23K	326K	1,035K	Y	601.59	4,119	40K	311K	1,006K	Y	21.94
4 hyper-periods													
nxt.ok1	396	14,014	57K	1,825K	5,816K	Y	1,305	2,393	71K	471K	1,610K	Y	70.59
nxt.bug1	398	14,014	57K	1,825K	5,816K	N	1,406	2,393	71K	471K	1,610K	N	73.27
nxt.ok2	388	14,156	60K	1,850K	5,849K	Y	1,382	2,447	73K	475K	1,618K	Y	67.08
nxt.bug2	405	14,573	71K	1,887K	5,978K	N	362	2,722	94K	485K	1,667K	N	77.30
nxt.ok3	405	14,573	71K	1,884K	5,964K	U	—	2,722	93K	466K	1,723K	Y	101.01
aso.bug1	421	14,942	81K	2,359K	7,699K	N	894	3,115	115K	726K	2,741K	N	143.52
aso.bug2	421	15,097	81K	2,359K	7,689K	N	773	3,205	116K	692K	2,438K	N	107.66
aso.ok1	418	14,946	80K	2,331K	7,590K	U	—	3,119	114K	620K	2,188K	Y	110.21
aso.bug3	445	16,024	113K	5,016K	16,162K	N	9,034	4,161	167K	1,406K	4,774K	Y	215.02
aso.bug4	444	16,055	108K	4,729K	15,141K	N	6,016	4,148	161K	1,271K	4,295K	N	168.22
aso.ok2	443	16,056	109K	4,734K	15,159K	U	—	4,149	162K	1,289K	4,360K	Y	200.25

Results: Self-Balancing Robot

		MONOSEQ						HARMONICSEQ					
Name		SAT Size				S	Time	SAT Size				S	Time
	OL	SL	GL	Var	Clause		(sec)	SL	GL	Var	Clause		(sec)
1 hyper-period													
nxt.ok1	396	2,158	12K	128K	399K	Y	21.22	2,378	17K	110K	354K	Y	4.22
nxt.bug1	398	2,158	12K	128K	399K	N	6.22	2,378	17K	110K	354K	N	4.36
nxt.ok2	388	2,215	12K	132K	410K	Y	11.16	2,432	18K	111K	356K	Y	4.69
nxt.bug2	405	2,389	15K	135K	422K	N	8.66	2,704	23K	114K	372K	N	5.81
nxt.ok3	405	2,389	15K	135K	425K	Y	14.46	2,704	23K	109K	358K	Y	5.71
aso.bug1	421	2,557	17K	167K	541K	N	12.05	3,094	29K	173K	568K	N	6.67
aso.bug2	421	2,627	17K	167K	539K	N	11.61	3,184	29K	165K	549K	N	6.71
aso.ok1	418	2,561	17K	164K	525K	Y	22.20	3,098	28K	147K	486K	Y	6.51
aso.bug3	445	3,118	24K	350K	1,117K	N	22.15	4,131	41K	341K	1,108K	Y	19.27
aso.bug4	444	3,105	23K	325K	1,027K	N	16.32	4,118	40K	307K	1,001K	N	10.83
aso.ok2	443	3,106	23K	326K	1,035K	Y	601.59	4,119	40K	311K	1,006K	Y	21.94
4 hyper-periods													
nxt.ok1	396	14,014	57K	1,825K	5,816K	Y	1,305	2,393	71K	471K	1,610K	Y	70.59
nxt.bug1	398	14,014	57K	1,825K	5,816K	N	1,406	2,393	71K	471K	1,610K	N	73.27
nxt.ok2	388	14,156	60K	1,850K	5,849K	Y	1,382	2,447	73K	475K	1,618K	Y	67.08
nxt.bug2	405	14,573	71K	1,887K	5,978K	N	362	2,722	94K	485K	1,667K	N	77.39
nxt.ok3	405	14,573	71K	1,884K	5,964K	U	—	2,722	93K	466K	1,723K	Y	101.01
aso.bug1	421	14,942	81K	2,359K	7,699K	N	894	3,115	115K	726K	2,741K	N	143.52
aso.bug2	421	15,097	81K	2,359K	7,689K	N	773	3,205	116K	692K	2,438K	N	107.66
aso.ok1	418	14,946	80K	2,331K	7,590K	U	—	3,119	114K	620K	2,188K	Y	110.21
aso.bug3	445	16,024	113K	5,016K	16,162K	N	9,034	4,161	167K	1,406K	4,774K	Y	215.02
aso.bug4	444	16,055	108K	4,729K	15,141K	N	6,016	4,148	161K	1,271K	4,295K	N	168.22
aso.ok2	443	16,056	109K	4,734K	15,159K	U	—	4,149	162K	1,289K	4,360K	Y	200.25

Related Work

Sequentialization of Concurrent Programs (Lal & Reps ‘08, and others)

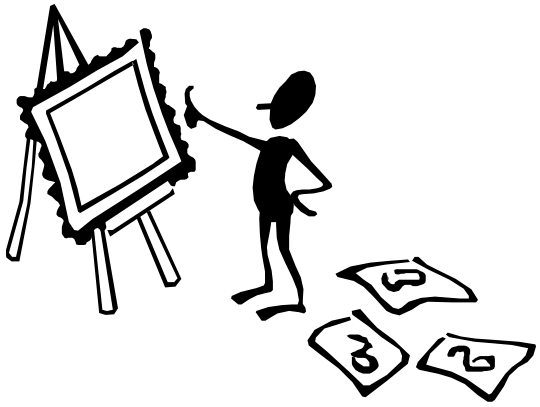
- Context Bounded Analysis of concurrent programs via sequentialization
- Arbitrary concurrent software
- Non-deterministic round robin scheduler
- Preserve executions with bounded number of thread preemptions
- Allow for arbitrary number of preemptions between tasks

Sequentialization of Periodic Programs (Kidd, Jagannathan, Vitek ’10)

- Same setting as this work
- Alternative sol’n: replace preemptions by non-deterministic function calls
- Additionally, supports recursion and inheritance locks
- No publicly available implementation – would be interesting to compare

Verification of Time Properties of (Models of) Real Time Embedded Systems

Conclusion



Past (FMCAD'11)

- Time Bounded Verification of Periodic C Programs
- Small (but hard) toy programs
- Reader/Writer protocols (with locks and lock-free versions)
- A robot controller for LEGO MINDSTORM from nxtOSEK examples

Present (VMCAI'13)

- Taking into account additional timing constraints for improved scheduling
  - arrival times, harmonicity, etc.
- A Lego Metal Stamping Robot (a.k.a. Turing Machine)
  - http://www.andrew.cmu.edu/~arieg/Rek (look for Turing Machine demo)

Current Work

- Verification without the time bound
- Back-End Verification engine
- Abstraction / Refinement
- Additional communication and synchronization
  - Priority-inheritance locks, message passing
- Modeling physical aspects (i.e., environment) more faithfully
- More Case studies and model problems

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0000142

QUESTIONS?

http://www.andrew.cmu.edu/~arieg/Rek

Sagar Chaki [chaki@sei.cmu.edu]