We are looking for **Post Doc Fellows** and **new Ph.D. Students** in Uppsala

Send me a message if you are interested

Wang Yi: yi@it.uu.se

---

**Schedulability Analysis of Timed Systems**

Wang Yi  
Uppsala University  

with contributions from  
Tobias Amnell, Elena Fersma, John Håkansson,  
Pavel Krčál, Leonid Mokrushin, and Paul Pettersson

Artist MOTIVES School, 2007, Trento, Italy

---

**Real Time Systems**

- Application software
- Scheduler (Resource management)
- Hardware
- RTOS
- Tasks
- Sensors
- Actuators

---

**Networks of Real-Time Components**  
(abstract view)

---

**Schedulability Analysis**

- Whether all task instances can be executed within given deadlines

- Alternatively (more difficult) what are the worst-case response times?

- (Buffer over- and under-flow? The buffer size?)

---

**The ABB Robot Controller**

- ABB robot controller (2 500 000 loc)
- Real time tasks A,B,C,D
- Read inputs from channels, write output to channels
- Task priority order D>C>B>A (PPS)
- Buffer overflow/underflow, WOR
Networks of Real-Time Components
(abstract view)

Timed automata with FIFO channels [CAV’06, Pavel&Wang]

PROBLEM SETTING

TASK -- a piece of executable code characterized by

- Worst-Case Execution time: C  (maybe [B, W])
- Priority: P
- Deadline: D
- Arrival Rate/pattern e.g. periodic

Scheduling Policy

- Decide which task to run
- e.g. EDF, FPS, FIFO, Rate-Monotonic etc.

Simple Task Arrival Patterns: Periodic

"Classic" Real Time Scheduling

Periodic tasks

well-developed techniques e.g. Rate-Monotonic Scheduling
In many applications:

- tasks may share many resources (not only CPU time)
- tasks may have complex control structures and interactions
- tasks may not be that "regular" (often non-periodic)

e.g. UML diagrams with SPT constraints

More Complex Arrival Patterns: Timed Traces

![Timed Trace Example]

(3.3, a), (3.4, b),
(6.5, a),
(3.6, a), (3.9, b),
(3.14, a), (3.14159, b)
...

Automata-based Approaches

Networks of timed automata whose transitions trigger tasks Pi’s

![Automata Diagram]

Scheduler/RTOS

Problem to solve:

\((A_1 || ... || A_n || \text{Scheduler}) \text{ satisfies } K\)

- \(K\) is a safety property (no deadline miss)
- \(\text{Scheduler}\) is an automaton encoding a given scheduling policy

OUTLINE

- A Model for Systems with Complex Task Structures [1998]
  - Timed automata with tasks
- Schedulability and (un)Decidability [Inf. & Com. 07]
  - Timed automata with bounded subtraction
- More Efficient Algorithms [TCS 06]
  - Schedulability analysis using 2 clocks
- TIMES tool demo
- Compositional Analysis: CATS tool [2007]
  - Keep the expressiveness for modeling
  - Perform analysis with approximation
- References/links

The MODEL

(Timed Automata with Tasks)
Timed Automata with Tasks

- Events
  - Discrete Transitions

- Timing constraints
  - Clocks / Guards / Resets
  - Complex arrival rates

- Tasks
  - Asynchronous execution
  - WCET, Deadline

Example: periodic tasks

\[ \text{start} \]
\[ c = 0 \]
\[ c \geq 100 \]
\[ x < 3 \]
\[ a! \]
\[ x := 0 \]

Timed Automata with Tasks

- Assume a set of tasks \( P_r \)
- A timed automaton with tasks is a tuple: \( <N,n_0,T,M> \)
  - \( N \) is a set of nodes
  - \( n_0 \) is the initial node
  - \( T \subseteq N \times (B(C) \times Act \times 2^C) \times N \) is the set of "edges"
    - \( C \) is a set of clocks
    - Act is a set of actions
    - \( B(C) \) is the set of clock constraints e.g. \( X < 10 \) etc.
  - \( M: N \rightarrow 2^{P_r} \) is a mapping which assigns each node a set of tasks

The Execution Platform

Thread 1

\[ P_3 \]
\[ P_2 \]
\[ P_1 \]
\[ P_2 \]

Task Queue

Thread 2: Scheduling Policy

Thread 3

Task Releases

\[ A_1 \parallel \ldots \parallel A_n \]

\[ \text{(the plant)} \]

The Execution Platform

Run of TAT

- (Idle, x=0, [\])
- 0.1 \rightarrow (Idle, x=0.1, [\])
- \rightarrow (RelP, x=0, [P(2,2)])
- 1.5 \rightarrow (RelP, x=1.5, [P(0,5.5),Q(2,2)])
- \rightarrow (RelQ, x=1.5, [Q(0,5.6.5)])
- 1.5 \rightarrow (RelQ, x=3, [Q(1,18.5)])
- \rightarrow (Idle, x=3, [Q(1,18.5)])
- \rightarrow (RelP, x=0, [P(2,8),Q(1,18.5)])
- 2 \rightarrow (RelP, x=2, [Q(1,16.5)])

States/Configurations of automata

A state is a triple: \((m, u, q)\)

Location (node) clock assignment (valuation) task queue
Sch and Run

- **Sch** is a function sorting task queues according to a given scheduler e.g. FPS, EDF, FIFO etc.

  Example: EDF \((P(2, 10), Q(4, 7)) = [Q(4, 7), P(2, 10)]\)

- **Run** is a function corresponding to running the first task of the queue for a given amount of time.

  Examples:
  - Run(0.5, [Q(4, 7), P(2, 10)]) = [Q(3.5, 6.5), P(2, 9.5)]
  - Run(5, [Q(4, 7), P(2, 10)]) = [P(1, 5)]

Semantics (as transition systems)

- **States:** \(<m, u, q>\)
  - \(m\) is a location
  - \(u\) is a clock assignment (valuation)
  - \(q\) is a queue of tasks (ready to run)

- **Transitions:**
  1. \((m, u, q) \xrightarrow{a} (n, r(u), \text{Sch}(M(n)::q))\) if \(g(u)\)
  2. \((m, u, q) \xrightarrow{d} (m, u+d, \text{Run}(d, q))\) where \(d\) is a real

OBS: \(q\) is growing (by actions) and shrinking (by delays)

"Zenoness" = Non-Schedulability

Schedulability of automata

A state is a triple: \((m, u, q)\)

- A state is schedulable if \(q\) is schedulable
- An automaton is schedulable if all reachable states are

Schedulability of Automata

Assume a scheduler \(\text{Sch}\):

- A state \((m, u, q)\) is schedulable with \(\text{Sch}\) if
  - \(\text{Sch}(q) = [P_1(c_1, d_1), P_2(c_2, d_2), \ldots, P_n(c_n, d_n)]\) and
  - \(c_1 + \ldots + c_i \leq d_i\) for all \(i \leq n\) (i.e., all deadlines met)
- An automaton is schedulable with \(\text{Sch}\) if all its reachable states are schedulable
- An automaton is schedulable with a class of scheduling policies if it is schedulable with every \(\text{Sch}\) in the class.
DECIDABILITY

Schedulability Analysis (Non-preemptive scheduling)

FACT [1998]
For Non-preemptive schedulers, the schedulability of an automaton can be checked by reachability analysis on ordinary timed automata.

Proof ideas (1):
Size of schedulable queues is bounded
- The maximal number of instances of \( P_i \) in a schedulable queue is bounded by \( M_i = \left\lfloor \frac{D_i}{C_i} \right\rfloor \)
- The maximal size of schedulable queues is bounded by \( M_1 + M_2 + \ldots + M_n \)
- To code the queue/scheduler, for each task instance, use 2 clocks:
  - \( c_i \) remembers the computing time
  - \( d_i \) remembers the deadline

\[
\begin{array}{c|c|c}
& (c_i, d_i) & \\
\hline
\end{array}
\]

Proof ideas (2):
The scheduler as an automaton

The scheduler automaton

Proof Ideas (3)
- Modify the original automaton \( M \): adding ‘release!’ to inform the scheduler
- Check reachability of the error state for \( M' \parallel \text{SCHEDULER} \)
How about preemptive scheduling?

- We may try the same ideas
  - Use clocks to remember computing times and deadlines
- BUT a running task may be stopped to run a more 'urgent' task
  - Thus we need stop-watches to remember "accumulated computing times"
- Then the schedulability problem is undecidable?
- This is wrong!!

Decidability Result [TACAS 2002]

FACT

For Preemptive schedulers, the schedulability of an
automaton can be checked by reachability analysis on
Bounded Subtraction Timed Automata (BSA).

NOTE

- Reachability for BSA is decidable
- Preemptive EDF is optimal; thus the general schedulability
  checking problem is decidable.

Timed automata with subtraction
i.e. Subtraction Automata, [McManis and Varaiya, CAV’94]

- Subtraction automata are timed automata
  extended with subtraction on clocks
- That is, in addition to reset x:=0, it is also
  allowed to update a clock
  x with x:= x-n where n is a natural number

Schedulability Checking as a reachability problem for
Bounded Subtraction Automata

- Model the scheduler as a subtraction automaton
  - Do not stop the computing clock c_i when a new task P_i is released
  - Let c_i, for P_i (preempted) run until the task P_i (with higher priority) finishes,
    then perform c_i:=c_i-C_i (note: c_i is the computing time for P_i).
Proof ideas (clocks are bounded):

- $c_2$ can never be negative.
- $c_i$ is bounded by $D_i$.

Schedulability analysis using DBM’s

Complexity

- #clocks (needed) = 2 x #instances (maximal number of schedulable task instances) = 2 x $\sum_i \frac{D_i}{C_i}$

This is a huge number in the worst case
But the run-time complexity is not so bad!

It works anyway !!!

- #active tasks in the queue is normally small, and the run-time complexity is only related to #active clocks
- If Too many active tasks in the queue (i.e. Too many active clocks), the check will stop sooner and report "non-schedulable"
- AND the analysis can be done symbolically!

WE CAN DO BETTER! [TACAS 03, TCS 06]

For fixed priority scheduling strategies (FPS),
we need only 2 clocks (and ordinary timed automata)!
The 2-CLOCK ENCODING
(for fixed-priority scheduler)

Problem to solve

A1 || A2 || ... || An || FPScheduler

Check: the network will never reach a state where a deadline is violated

Main Idea

- Check the schedulability of tasks one by one according to priority order (highest priority first)
- This is similar to response time analysis in RMS

To code the scheduler, we need:

- 1 integer variable for each Pi:
  - r denotes the response time as in RMS (the total computing time needed before Pi finishes)
- 2 clocks for each Pi:
  - c remembers the accumulated computing time (so much has been computed so far)
  - d remembers the "deadline"

Intuition of the encoding:

\[ R_i = C_i + \sum_{\text{prior}(P_j) > \text{prior}(P_i)} C_j \]

- Assume: priority(Pi) > priority(Pj) and Pi is analyzed

When Pi finishes, \( r = R_i \)

Note that it is not clear that c and r are bounded!
The “FPS scheduler”: analyzing Pi (we need the boundedness)

- \( c \) and \( r \) are bounded
  - \( c \) is bounded by \( M \)
  - \( r \) is bounded by \( r_{\text{max}} + C_i \)
    - Where \( r_{\text{max}} \) is the maximal value of \( r \) from previous analysis for all tasks \( P_j \) with higher priority

So the scheduler is a standard TA

SUMMARY: Decidability

- For non-preemptive schedulers, the problem can be solved using standard TA.
- For preemptive schedulers, the problem can be solved using BSA (Bounded Substraction Automata).
- For fixed-priority schedulers, the problem can be solved using TA with only 2 extra clocks – similar to the classic RMA technique (Rate-Monotonic Analysis).

Undecidability [Inf. and Comp. 2007]

Unfortunately, the problem will be undecidable if the following conditions hold together:

1. Preemptive scheduling
2. Interval computation times \([B, W]\)
3. Feedback i.e. the finishing time of tasks may influence the release times of new tasks.

What we have done so far:

Compositional Analysis of Timed Systems with Abstraction/Approximation

www.timestool.com/cats
The ABB Robot Controller

\[ \text{Task Ready Queue} \]

\[ \text{Shared variables} \]

\[ \text{TAA\times TAB\times TAC\times TAD\times TASCH with queues is TOO BIG} \]

UPPAAL, TIMES (and others)

Trying to search "all the combinations of local states":

\[ S_1 || S_2 || ... || S_m || q_1 || q_2 || ... || q_n \]

Some of which are bugs

Networks of RT Components

Components: A1, A2 ... Am and queues: Q1, Q2 ... Qn

System/component = Stream transformer

- Network Calculus (Cruz, Boudec, Thiran ’91-’04)
  - Arrival Curves
- Real-Time Calculus (Thiele, Chakraborty ’00s)
  - Upper/Lower Arrival/Service Curves
- RT Components/TA = Abstract stream transformers
  - Abstract stream defines a timed language

Set of streams = Abstract stream = Arrival curve

a sliding time window with size 5

Set of streams = Abstract stream = Arrival curve

a sliding time window with size 5
Set of streams = Abstract stream = Arrival curve

A sliding time window with size 5

---

Set of streams = Abstract stream = Arrival curve

A sliding time window with size 5

---

Set of streams = Abstract stream = Arrival curve

A sliding time window with size 5

---

Set of streams = Abstract stream = Arrival curve

A sliding time window with size 5

---

Set of streams = Abstract stream = Arrival curve

A sliding time window with size 1

---

Arrival curve

\[ L(C) = \text{set of streams} \]
System/component = “Arrival curve” transformer

This can be done modularly when there is no “feedback”

TA as Curve Transformer

System/Component = Arrival Curve Transformer

• Comparing the curves we will answer:
  - if A1 and A2 can “work together”? (all the events generated by A1 will be received and processed by A2)
  - what is the sufficient size of the buffer?
  - what is the output curve of A2?

How about resources and scheduling?

Real-Time Calculus, Lottar et al

Arrival Curves (tasks) Service Curves (resources)

(a,3),(a,3.34),(a,3.39),(a,4),(a,10)... (100%,0),(50%,3.3),(100%,7)...

Properties of Curves

• max vertical distance = required buffer size
• max horizontal distance = flow delay bound
Resources & Scheduling

- FPS, priority order:
  - Priority(A) < Priority(B) < Priority(C) < Priority(D)
- Service Curves
  - Same as arrival curves but express available resource within windows
  - Highest priority task has 100% of CPU

Networks of Real-Time Components
(abstract view)

An Example with Feedback

- TASK1 input depends on the TASK2 output
- TASK1 uses TASK2's remaining resource
- TASK2 input depends on TASK1 output
- Given
  - TASK1 input stream
  - Initial condition on activation of TASK2
- Iterative computation until fixed point

Simple Scheduling Example

- 4 tasks: 3 periodic + 1 aperiodic (TA)
- Preemptive fixed priority scheduling
- Given BCET/WCET
- Abstracting release pattern with streams
- Analysis
  - Worst case response time
  - Required OS ready queue size

References/links

- TIMES: www.timestool.com
- CATS: www.timestool.com/cats
- UPPAAL: www.uppaal.com