OUTLINE

- Introduction
  - Lecture 1: Motivation, examples, problems to solve
- Modeling and Verification of Timed Systems
  - Lecture 2: Timed automata, and timed automata in UPPAAL
  - Lecture 3: Symbolic verification: the core of UPPAAL
  - Lecture 4: Verification Options in UPPAAL
- Towards a Unified Framework
  - Lecture 5: Modeling, verification, real time scheduling, code synthesis
    From UPPAAL to TIMES

Unification of Scheduling, Model-Checking and Code Synthesis: From UPPAAL to TIMES

"Who is Who" in Real Time Systems

- Real Time Scheduling (RTSS ...)
  - Task models, Schedulability analysis
  - Real time operating systems
- Automata/logic-based methods [CAV, TACAS ...]
  - FSM, PetriNets, Statecharts, Timed Automata
  - Modeling, Model checking ...
- (RT) Programming Languages [...]}
  - Esterel, Signal, Lustre, Ada ...
  - ...

The Same Goal: Reliable Controllers
(with minimal resource consumption)

The main components of a controller a set of tasks: P1, P2 ... Pn running on a platform (RTOS: scheduler)

The design problem

- A set of computation tasks
  - Timing constraints: e.g. Deadlines
  - (QoS constraints: 80% of deadlines met, liveness?)
  - Release patterns i.e Task models
- Design a controller/Schedule
  - To ensure the constraints

"Classic" Real Time Scheduling

- Periodic tasks
  - well-developed techniques e.g. Rate-Monotonic Scheduling
Rate-Monotonic Scheduling

- P1...Pn arrive at fixed rates
- Fixed Priority Order: higher frequency => higher priority
- Always run the task with highest priority (FPS)
  \[ P_1 || P_2 || \ldots || P_n || FPS \]
- Schedulability can be tested by utilization bound (or equation solving)

In real life, tasks may

- share many resources (not only CPU time)
- have complex control structures and interactions
- have to satisfy mixed logical, temporal & resource constraints

Automata-based Approaches

A controller = a set of timed automata accepting tasks P_i's

The TIMES project

Tools for Modeling and Implementation of Embedded Systems

Uppsala University

Vision

- Timed Model to Executable Code
  Guaranteeing Timing Constraints

- Timing analysis of Concurrent and Time-Critical Software
  - Response time estimation

Why this work?

- Timed Automata, [Alur&Dill 1990]
  - generated a lot of work on model checking:
    \[ P \models \varphi \]
  - Consider \( M \) as a design specification
  - Construct a program \( P \) from \( M \)
An Overview of TIMES

OUTLINE
- A Unified Model for Timed Systems [1998]
  - Timed automata with tasks
- Schedulability and Decidability [TACAS 02]
  - Timed automata with bounded subtraction
- More Efficient Algorithms [TACAS 03]
  - Schedulability analysis using 2 clocks
    - (similar to Rate-Monotonic Scheduling)
- Undecidability [TACAS 04]
  - The execution times of tasks are intervals
  - Task completion times influence task release times
- TIMES demo

Implemented in the TIMES tool

The MODEL
(Timed Automata with Tasks)

Modelling Real Time Systems
- Events
  - synchronization
  - interrupts
- Timing constraints
  - specifying event arrivals
  - e.g. Periodic and sporadic
- Tasks (executable programs)
  - interrupt processing
  - Internal computation
    - triggered by events and scheduled in the ready queue of RTOS
  - (Timed Automata + tasks)

Example: periodic tasks

Tasks = Executable Programs (e.g. C, Java)
- Task parameters:
  - C: WCET
  - D: Relative Deadline
  - (other parameters for scheduling e.g. Priority)
- Task Interface:
  Task P
    \[
    \begin{align*}
    v_1 & := f_1(x_1, \ldots, x_n) \\
    \vdots & \\
    v_n & := f_n(x_1, \ldots, x_n) \\
    \end{align*}
    \]
  (a set of variables updated)
Timed Automata with Tasks (Example)

- Processor 1 (event handler)
  - Initially, P in the queue
  - Run-to-Completion/Stabilization
    - Whenever a available and x>10, Q is put in the queue
    - Then
      - Whenever b available and y<=50, P is put in the queue
  - Processor 2 (task handler)
    - Schedule and Compute tasks in the queue

Example transitions:
- \((P, x=0, y=0, [P(1,7)]) \rightarrow (P, x=0.6, y=0.6, [P(0.4, 6.4)])\)
- \((P, x=0, y=0, [P(1,7)]) \rightarrow (Q, x=10.1, y=0, [Q(3,9)])\)
- \((R, x=10, y=2, [Q(3,7), Q(3,9)]) \rightarrow (R, x=12.1, y=2, [Q(3,7)])\)
- \((Q, x=10.1, y=0, [Q(3,7), Q(3,9)]) \rightarrow (R, x=12.1, y=2, [Q(3,7)])\)
- \((R, x=10, y=2, [Q(3,7), Q(3,9)]) \rightarrow (R, x=12.1, y=2, [Q(3,7)])\)

States/Configurations of automata

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

Example

Initial State: \((P, x=0, [P(1,7)])\)

Example transitions:
- \((P, x=0, [P(1,7)]) \rightarrow (P, x=0.6, [P(0.4, 6.4)])\)
- \((P, x=0, [P(1,7)]) \rightarrow (Q, x=10.1, y=0, [Q(3,9)])\)
- \((Q, x=10, y=2, [Q(3,7), Q(3,9)]) \rightarrow (R, x=12.1, y=2, [Q(3,7)])\)
- \((R, x=10, y=2, [Q(3,7), Q(3,9)]) \rightarrow (R, x=12.1, y=2, [Q(3,7)])\)

We need to handle the queue by Run and Sch
**Sch and Run**

- **Sch** is a function sorting task queues according to a given scheduling strategy 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: \( \text{Run}(0.5, [Q(4, 7), P(2, 10)]) = [Q(3.5, 6.5), P(2, 9.5)] \)

  \( \text{Run}(5, [Q(4, 7), P(2, 10)]) = [P(1, 5)] \)

**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. \( (m,u,q) \rightarrow (n, r(u), \text{Sch}[M(n)::q]) \) if \( &g(u) \)
  2. \( (m,u,q) \rightarrow (m, u+d, \text{Run}(d,q)) \) where \( d \) is a real

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

---

**Zenoness = Non-Schedulability**

- Start state: \( P=(2,3) \)

  Zenoness: \( \infty \) many \( P \)'s may arrive within 1 time unit!

  But after 2 copies, the queue will be non-schedulable

---

**Schedulability of Automata**

- A state is a triple: \( (m, u, q) \)
  - Location
  - Clock assignment
  - Task queue

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

---

**Schedulability of Automata**

- Assume a scheduling policy \( \text{Sch} \):
  - A state \( (m,u,q) \) is schedulable with \( \text{Sch} \) if
    - \( \text{Sch}(q)= [P_1(c_1,d_1), P_2(c_2,d_2), \ldots, P_n(c_n,d_n)] \)
    - \( c_1 + \ldots + c_i \leq d_i \) 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.
Other verification/scheduling problems

- Location Reachability (just as for timed automata)
  - a nice property of the model!
- Boundedness of the task queue \( |q| < M \)
  - memory requirement
- Schedule synthesis

---

**DECIDABILITY**

---

Schedulability Analysis (Non-preemptive scheduling)

**FACT** [1998]

For Non-preemptive scheduling strategies, 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

---

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 $M' || SCHEDULER$

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 computing times

Conjecture (1998 @ Grenoble, VHS meeting):

- The schedulability problem for Preemptive scheduling is undecidable.

- The intuition: we need stop-watch to code the scheduler and the reachability problem for stop-watch automata is undecidable

- This is wrong !!!

Decidability Result [TACAS 2002]

FACT

For Preemptive scheduling strategies, 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

Bounded Subtraction Automata

- A subtraction automaton is bounded if its clocks are non-negative and bounded with a maximal constant (or subtraction is only allowed in the bounded zone).

FACT:

Location Reachability checking is decidable!
Schedulability Checking as a reachability problem for Bounded Subtraction Automata

Proof ideas (no stop but subtraction :-)
- Model the scheduler as a subtraction automaton
  - Do not stop the computing clock $c_2$ when a new task $P_1$ is released
  - Let $c_2$ for $P_2$ (preempted) run until the task $P_1$ (with higher priority) finishes, then perform $c_2 := c_2 - C_1$ (note: $C_1$ is the computing time for $P_1$).

Proof ideas (clocks are bounded):
- $c_2$ can never be negative.
- $c_2$ is bounded by $D_2$.

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

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

It works anyway !!!
- $\#\text{active tasks}$ in the queue is normally small, and the run-time complexity is only related to $\#\text{active clocks}$
- 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!
Schedulability analysis based on Constraints (DBM’s)

Subtraction on Clocks, added to DBM-library (UPPAAL, Kronos)

WE CAN DO BETTER! [TACAS 03]

For fixed priority scheduling strategies (FPS), we need only 2 clocks (and ordinary timed automata)

The 2-CLOCK ENCODING
(for fixed-priority scheduling strategies)

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 queue/scheduler, we need:

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

Intuition of the encoding: $R_i = C_i + \sum_{pri(P_j) > pri(P_i)} C_j$

– Assume: priority(P_j) > priority(P_i) and P_i is analyzed

When P_i finishes, $r = R_i$
The “FPS scheduler”: analyzing Pi

The “FPS scheduler”: analyzing Pi (we need the boundedness)

Note that it is not clear that c and r are not bounded!

\[
d:=0 \\
r:=r+C_j \\
c:=0, r:=C_j \\
c=r, d=Di \\
c<r, d=Di \\
r:=r+C_j \\
c:=M \\
c:=0 \\
r:=r-M \\
OBS: r-c is the only interesting info, so M can be any integer! Let M=C_i
\]

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

An Overview of TIMES

The INPUT LANGUAGE is very much like "guarded commands"

- guard, update: "synchronous" computation which takes "no time" – we adopt the synchronous hypothesis
- task: "asynchronous" computation which takes time
Tasks = Executable Programs (e.g. C, Java)

- Task Type
  - Synchronous or Asynchronous
  - Non-Periodic (triggered by events) or Periodic
- Task parameters: C, D etc.
  - C: Computing time and D: Relative Deadline
  - other parameters for scheduling e.g. priority, period
- Task Interface (variables updated ‘atomically’)
  - X := f(x1...xn)
- Tasks may have shared variables
  - with automata
  - with other tasks (priority ceiling protocols)
- Tasks with Precedence constraints

Functionality/Features of TIMES

- GUI
  - Modeling: automata with (a)synchronous tasks
  - editing, task library, visualization etc
- Simulation
  - Symbolic execution as MSC’s and Gantt Charts
- Verification
  - Safety, bounded liveness properties (all you do with UPPAAL)
  - Schedulability analysis
- Synthesis
  - Verified executable code (guaranteeing timing constraints)
  - Schedule synthesis (ongoing)

CODE SYNTHESIS in TIMES

- Run Time Systems
  - Event Handler
    - OS interrupt processing system or Polling
  - Task scheduler
    - generated from task parameters
- Application Tasks = threads (or processes)
  - Already there! (written in C)
  - Current version of TIMES support LegoOS!

TIMES

Conclusions/Remarks

- A unified model for timed systems (can express synchronization, computation and complex temporal and resource constraints).
- The first decidability result (and efficient algorithms)
  for preemptive scheduling in dense time models:
  - The analysis is symbolic (using DBMs in the UPPAAL tool)
  - The results can be adopted for schedulability analysis of
    message transmission.
- Implementation: TIMES