

OUTLINE

- A Brief Introduction
 - Motivation ... what are the problems to solve
 - CTL, LTL and basic model-checking algorithms
- Timed Systems
 - Timed automata and verification problems
 - UPPAAL tutorial (1): data structures & algorithms
 - UPPAAL tutorial (2): input languages
 - **TIMES**: From models to code "guaranteeing" timing constraints
- Further topics/Recent Work
 - Systems with buffers/queues [CAV 2006]

Lecture 8

Unification of Model-Checking, Scheduling, and Code Synthesis:

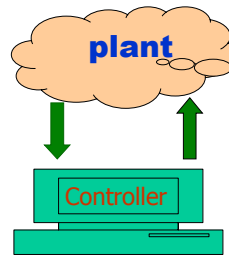
From UPPAAL to TIMES

"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
 - Modelling, Model checking ...
- (RT) Programming Languages [...]
 - Esterel, Signal, Lustre, Ada ...
-

The Same Goal: Reliable Controllers

(with minimal resource consumption)



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

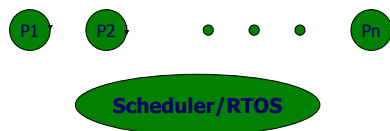
P1 || P2 || ... || Pn || Scheduler

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

- $P_1 \dots P_n$ arrive at **fixed rates**
- Fixed Priority Order: **Higher frequency => Higher priority**
- Always run the task with highest priority (FPS)
 $P_1 \parallel P_2 \parallel \dots \parallel P_n \parallel \text{FPS}$
- Schedulability can be tested by utilization bound (or equation solving)

7

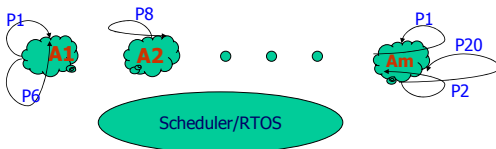
In real life, tasks may

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

8

Automata-based Approaches

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



9

The TIMES project

Tools for **M**odeling and **I**mplementation of **E**mbedded **S**ystems

Uppsala University

10

Vision

- **Timed Model to Executable Code**
Guaranteeing Timing Constraints
- **Timing analysis of Concurrent and Time-Critical Software**
 - Response time estimation

11

Problems to solve

- **Schedulability analysis:** check
 $(A_1 \parallel A_2 \parallel \dots \parallel A_n \parallel \text{Scheduler})$ satisfies K
 - A scheduler is given e.g. FPS, RMS, EDF etc
 - K is a requirement specifying e.g. safety & liveness
- **Schedule synthesis:** find X such that
 $(A_1 \parallel A_2 \parallel \dots \parallel A_n \parallel X)$ satisfies K

All these can be automated

12

OUTLINE

- A 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

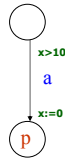
Implemented in the **TIMES** tool

The MODEL

(Timed Automata with Tasks)

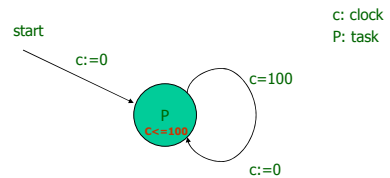
Modelling Real Time Systems

- 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 Automaton + 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)

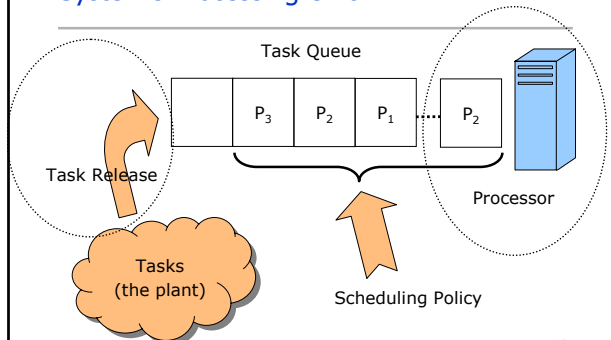
Task Interface:

```

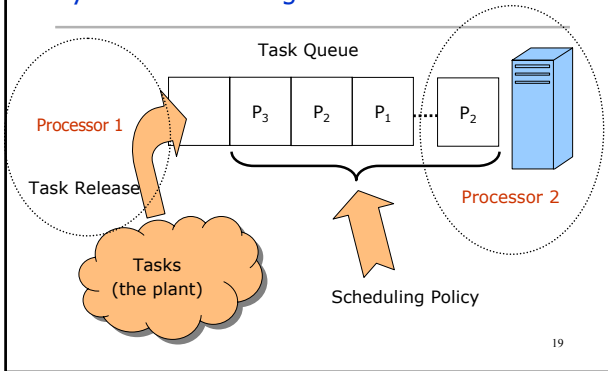
Task P
{
  v1 := F1(v1...vn)
  ...
  vn := Fn(v1...vn)
}
    
```

(a set of variables updated)

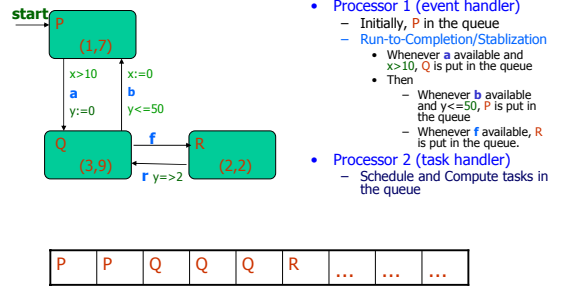
System's Processing Unit



System's Processing Unit



Timed Automata with Tasks (Example)



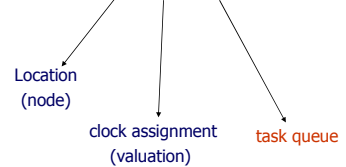
Timed Automata with Tasks [1998]

- Assume a set of tasks Pr
- A timed automaton with tasks is a tuple: $\langle N, n_0, T, M \rangle$
 - $\langle N, n_0, T \rangle$ is a standard timed automaton
 - 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
 - $M: N \rightarrow 2^{Pr}$ is a mapping which assigns each node a set of tasks

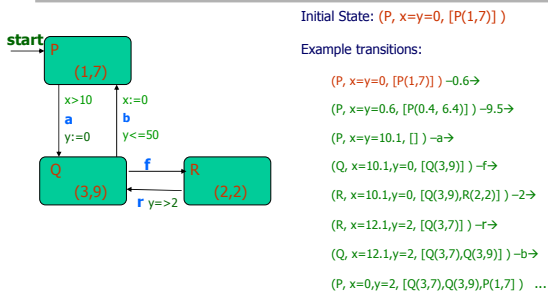
21

States/Configurations of automata

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



Example



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: $Run(0.5, [Q(4, 7), P(2, 10)]) = [Q(3.5, 6.5), P(2, 9.5)]$
 $Run(5, [Q(4, 7), P(2, 10)]) = [P(1, 5)]$
- 24

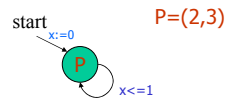
Semantics (as transition systems)

- States: $\langle m, u, q \rangle$
 - 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 $(m \xrightarrow{g} n) \& g(u)$
 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)

25

Zenoness = Non-Schedulability



Zeno: ∞ many P 's may arrive within 1 time unit !



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

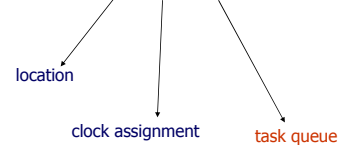
26

SCHEDULABILITY

27

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

28

Schedulability of Automata

Assume a scheduling policy Sch :

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

29

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

30

DECIDABILITY

Schedulability Analysis (Non-preemptive scheduling)

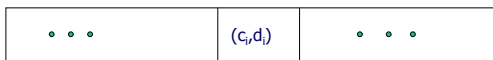
FACT [1998]

For Non-preemptive scheduling strategies, the schedulability of an automaton can be checked by reachability analysis on ordinary timed automata.

Proof ideas (1):

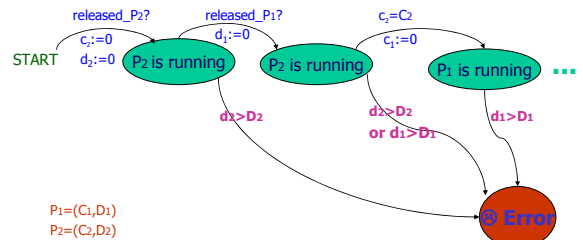
Size of schedulable queues is bounded

- The maximal number of instances of P_i in a schedulable queue is bounded by $M_i = \lceil D_i / C_i \rceil$
- The maximal size of schedulable queues is bounded by $M_1 + M_2 + \dots + M_n$
- To code the queue/scheduler, for each task instance, use 2 clocks:
 - c_i remembers the computing time
 - d_i remembers the deadline

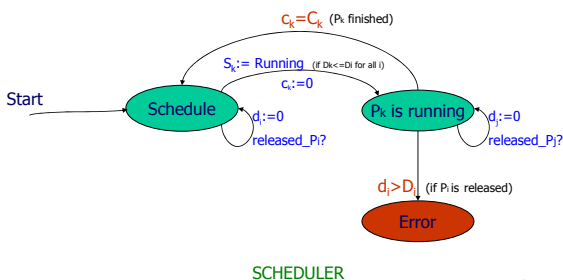


Proof ideas (2):

The scheduler as an automaton

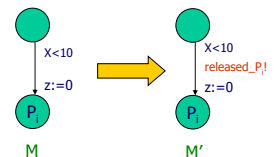


The scheduler automaton



Proof Ideas (3)

- Modify the original automaton M : adding 'release!' to inform the scheduler



- Check reachability of the error state for $M' \parallel \text{SCHEDULER}$

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 computing times

37

Conjecture (1998):

- 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 !!!

38

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

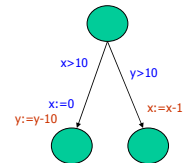
- Reachability for BSA is decidable
- Preemptive EDF is optimal; thus the general schedulability checking problem is decidable.

39

Timed automata with subtraction

i.e. Subtraction Automata, [McManis and Varaiya, CAV94]

- Subtraction automata are timed automata extended with subtraction on clocks

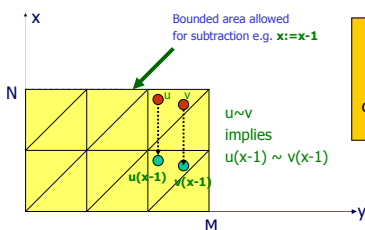


- That is, in addition to reset $x := 0$, it is also allowed to update a clock x with $x := x - n$ where n is a natural number

40

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).



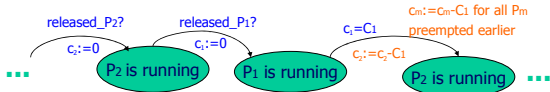
FACT:
Location Reachability checking is decidable!

41

Schedulability Checking as a reachability problem for Bounded Subtraction Automata

42

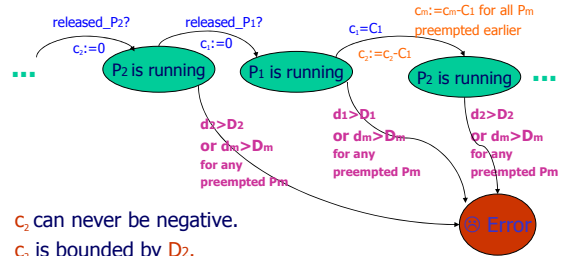
Proof ideas (no stop but subtraction :-)



- 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_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).

43

Proof ideas (clocks are bounded):



- C_2 can never be negative.
- C_2 is bounded by D_2 .

44

END of proof

45

Complexity

$$\begin{aligned} \# \text{clocks (needed)} &= 2 \times \# \text{instances (maximal number of schedulable task instances)} \\ &= 2 \times \sum_i \lceil D_i / C_i \rceil \end{aligned}$$

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

46

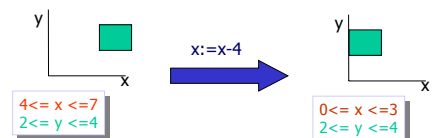
It works anyway !!!

- #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!

47

Schedulability analysis based on Constraints (DBM's)

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



48

WE CAN DO BETTER ! [TACAS 03]

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

49

The 2-CLOCK ENCODING

(for **fixed-priority** scheduling strategies)

50

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**

51

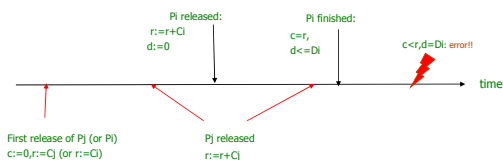
To code the queue/scheduler, we need:

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

52

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

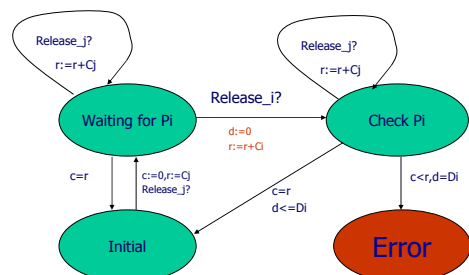
– Assume: $\text{priority}(P_j) > \text{priority}(P_i)$ and P_i is analyzed



When P_i finishes, $r = R_i$

53

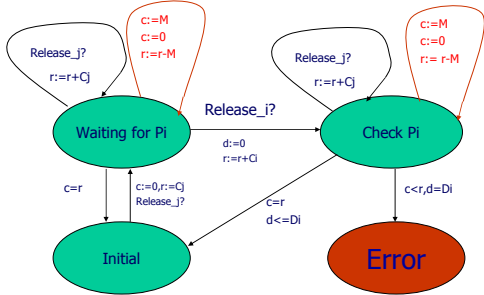
The "FPS scheduler": analyzing P_i



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

54

The "FPS scheduler": analyzing P_i (we need the boundedness)



OBS: $r \cdot c$ is the only interesting info, so M can be any integer! Let $M=C$

55

c and r are bounded

- c is bounded by M
- r is bounded by $\Gamma_{\max} + C_i$
 - Where Γ_{\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 **END**

56

Decidability results

- For Non-preemptive scheduling, the problem can be solved using TA.
- For preemptive scheduling, the problem can be solved using BSA (Bounded Substraction Automata) [TACAS02]
- For fixed-priority scheduling, the problem can be solved using TA with only 2 extra clocks – similar to the classic RMA technique (Rate-Monotonic Analysis) [TACAS03]

57

Undecidability [TACAS 04]

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

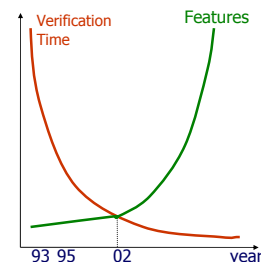
- Preemptive scheduling
- Interval computation times
- Feedback i.e. the finishing time of tasks may influence the release times of new tasks.

58



59

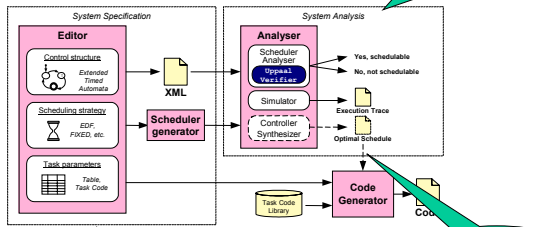
The past and future of UPPAAL



- Commercial Tools
 - focus: code generation
 - new features ...
- Academic Tools
 - focus: modeling & verification
 - new engines ...
- Verification to Synthesis !**

60

An Overview of **TIMES**



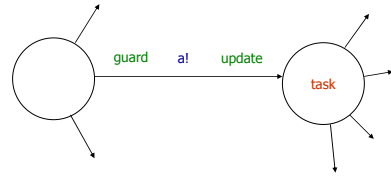
Analysis

Modeling

Synthesis

The **INPUT LANGUAGE** is very much like **"guarded commands"**

OBS: **guard** and **update** may contain data variables (integer, array)



- **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')
 - $X_i := F(X_1, \dots, X_n)$
- Tasks may have shared variables
 - with automata
 - with other tasks (priority ceiling protocols)
- Tasks with Precedence constraints

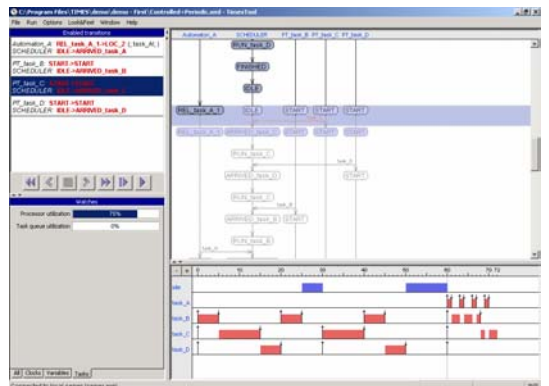
Functionality/Features of **TIMES**

- GUI
 - Modeling: automata with (a)synchronous tasks
 - editing, task library, visualization etc
- Simulation
 - Symbolic execution as MSC's and Gant Charts
- Verification
 - Safety, bounded liveness properties (all you do with UPPAAL)
 - Schedulability analysis
- Synthesis
 - Verified executable code (guaranteeing timing constraints)
 - Traces(Code) \subseteq Traces(Model)
 - 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 demo



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 DBM's in the UPPAAL tool)
 - The results can be adopted for schedulability analysis of message transmission.
- **Implementation: TIMES**