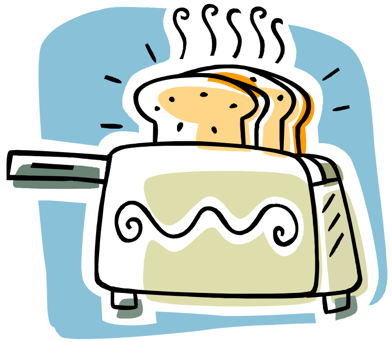
Model-Checking, Scheduling Analysis (and Code Synthesis): Times

Alexandre David
1.2.05
adavid@cs.aau.dk

Thanks to Wang Yi

Classical approach to RTS

- Decompose the controller as
  - a set of tasks *computations*
  - running on a RTOS *scheduler*
- Constraints:
  - timing – deadlines
  - QoS
  - task model – release pattern

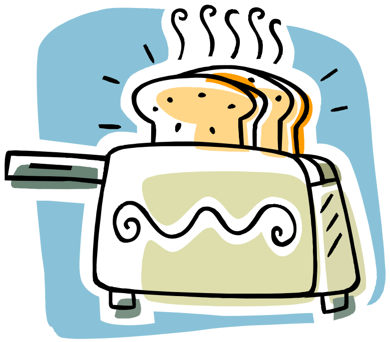


heater
timer
temperature monitor
security switch
anti-bread-burning
…

How to get it right?

How to get a correct controller?

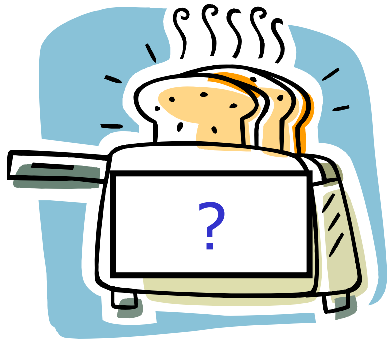
Verification -
Model-checking



UPPAAL

Is my system correct?
Does it satisfy its
requirements?

Code synthesis



TIGA

Generate the code
for a correct controller.

A bit of both: Check design – schedulability,
generate scheduler, put together the tasks.

Times

Research directions

- 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 ...

Motivation

- Classic RTS scheduling:
  - define tasks, computation time C, period T, deadline D, assign priority P
  - different scheduling policies
    - fixed: rate monotonic (T), deadline monotonic (D)
    - dynamic: EDF (D)
  - analytical solving
- But in practice tasks have
  - shared resources
  - dependencies
  - complex control structures & interactions

Wish List

- From a timed model to executable code.
  - Generated $\rightarrow$ guarantee correctness *dependencies, timing, shared resources*…
- Timing analysis of RTS.
  - Different scheduling policies.
  - WCRT

Approach with Times

- Use TA to model the arrival pattern of tasks.
  - Have default policies included for convenience.
- Augment the model with a scheduler.
  - And shared resources + dependencies.

- Check for schedulability using UPPAAL as the back-end model-checker.
- Generate code of the scheduler (with custom arrival pattern).

Problem Statement

- Schedulability analysis
  - $(A_1 \,||\, A_2 \,||\, .. \, A_n \,||\, \text{Scheduler}) \models \phi$?
  - Scheduler given with a policy.
  - ϕ is a requirement – formula in some logic.

UPPAAL

Times

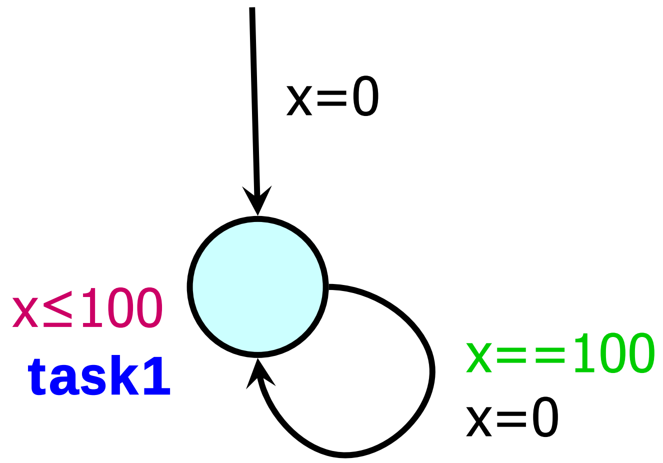
- Schedule synthesis
  - Find X s.t. $(A_1 \,||\, A_2 \,||\, .. \, A_n \,||\, X) \models \phi$

TIGA

Modeling

- RTS behavior: TA.
  - General approach, general model-checker.
- Schedulability analysis: TA + tasks.
  - Add tasks to the model.
  - TA used to model the task arrival pattern.
  - Idea: any pattern available, with any kind of dependency, including resource sharing.

Example: Periodic Task



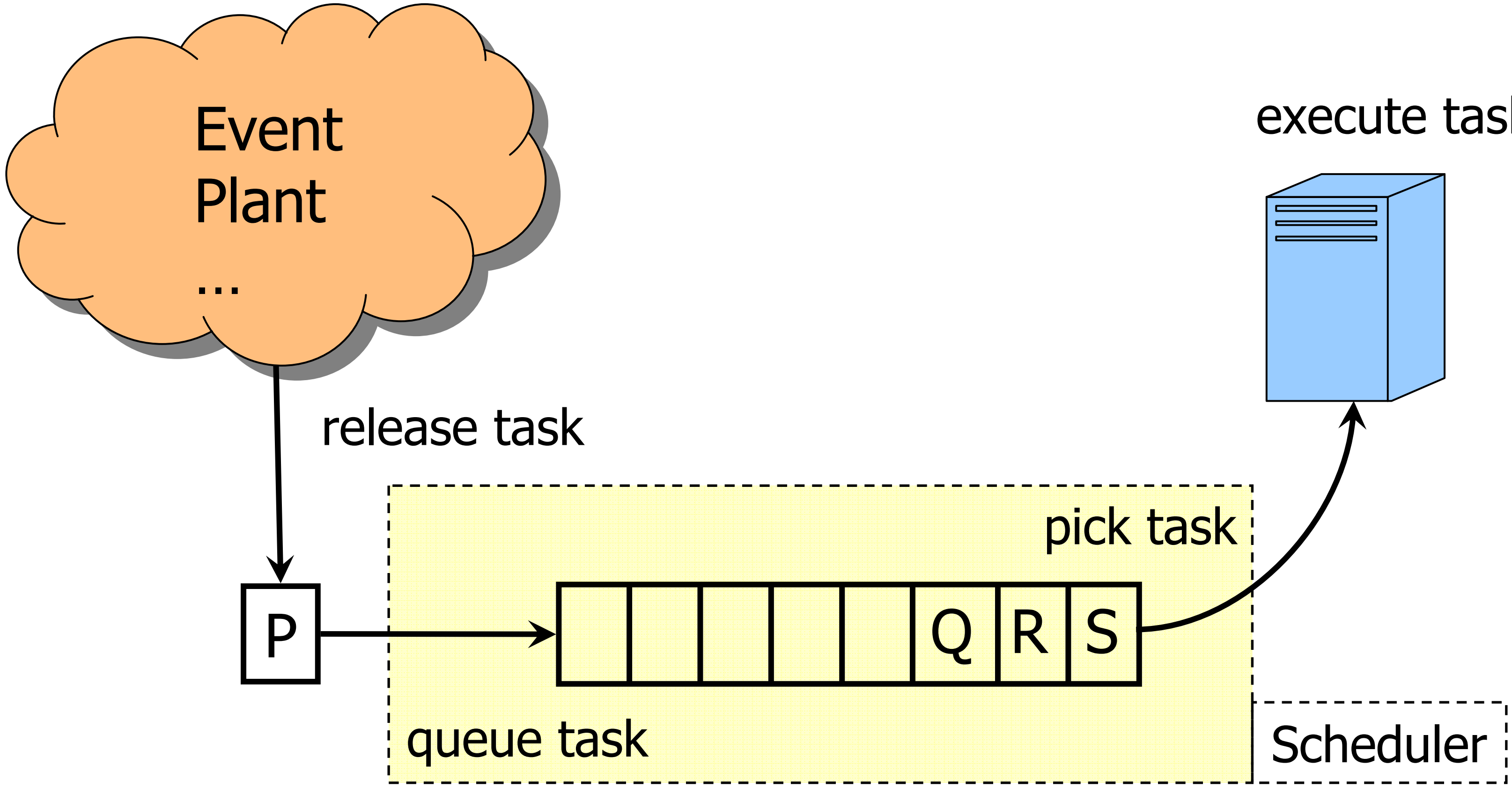
Whenever you enter that location,
release task1.
Model → every 100 time units.

Modeling with Tasks

- From a modeling point of view
  a task = some external program.
  - Can interact with the model through an interface.

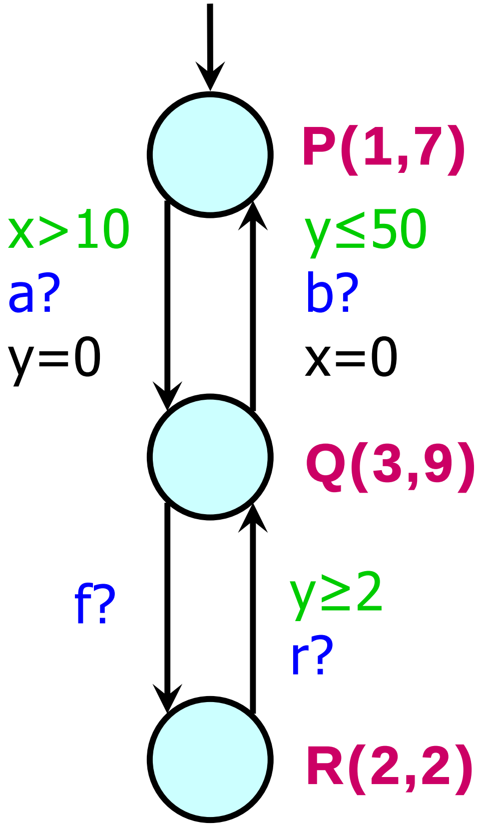
- Parameters:
  - WCET
  - Deadline
  - *Period*
  - Dependencies
  - Resource access

Model of the System Execution



How to queue & pick a task:
Scheduling policy.

TAT Example



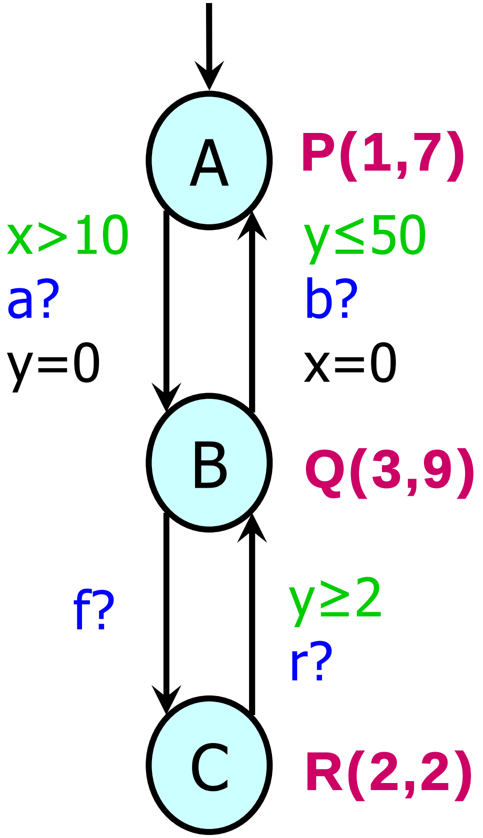
Task(C,D)

- Event handler:
  - Release P initially.
  - Run-to-completion semantics:
    - whenever a? and x>10, release Q
    - then whenever b? and y≤50, release P, or whenever f, release R
    - …
- Task handler
  - schedule & compute tasks

What is a TAT?

- Take a TA <L, l_0, T, I>
  - Locations, initial location, Transition relation, Invariants.
- Add a mapping $M: L \rightarrow 2^P$ with P being a set of tasks.
- Semantics
  - TA states: (l,v)
    location vector + clock valuations
  - TAT states: (l,v,q)
    … + task queue

TAT Example



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

Example transitions:

delay 0.6 → (A, x=y=0.6, [P(0.4,6.4)])
delay 9.5 → (A, x=y=10.1, [])
action a → (B, x=10.1,y=0, [Q(3,9)])
action f → (C, x=10.1,y=0, [Q(3,9),R(2,2)])
delay 2 → (C, x=12.1,y=2, [Q(3,7)])
action r → (B, x=12.1,y=2, [Q(3,7),Q(3,9)])
action b → (A, x=0,y=2, [Q(3,7),Q(3,9),P(1,7)])
...

Semantics

- $(l,v,q) \rightarrow (l',v',q')$ by 2 kinds of transitions:
  - actions: tasks may be added, q grows
    $(l,v,q) \rightarrow^{g,a,r} (l',v', Sch(M(l'),q))$ if g
  - delay: tasks are executed, q shrinks
    $(l,v,q) \rightarrow^{d} (l,v+d, Run(d,q))$ if $I(l)(v+d)$
  - Sch & Run: functions to update the queue.
    Sch: scheduling policy.
    Run: execute the first task.

Schedulability

- Bound instances of tasks.
- Bound the queue.
- Check that the queue is schedulable
  - stays within bounds
  - all deadlines are met

    A state (m,u,q) is schedulable with Sch if (given $Sch(q)= [P_1(c_1,d_1)P_2(c_2,d_2)\ldots P_n(c_n,d_n)]$) $(c_1+\ldots+c_i)<=d_i$ for all $i \leq n$.

Decidability Results

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

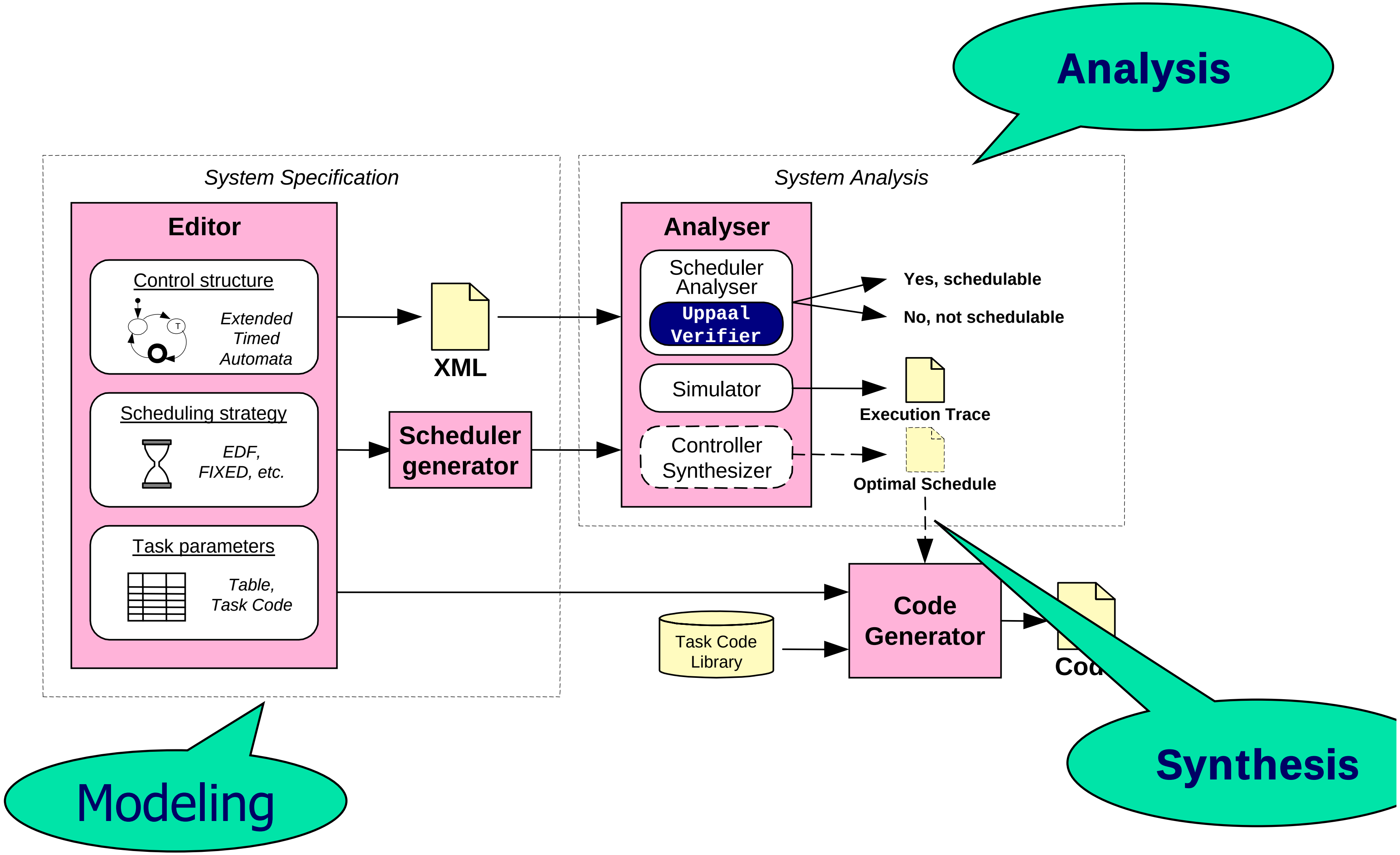
- [TACAS 2002]
  For Preemptive scheduling strategies, the schedulability of an automaton can be checked by reachability analysis on Bounded Subtraction Timed Automata (BSA).
  - Natural coding: Stop time when you preempt
    $\rightarrow$ stop-watches $\rightarrow$ undecidable.
  - Alternative: Use subtraction to “cancel” non-executed time.

- [TACAS 2003]
  For fixed-priority scheduling, the problem can be solved using TA with only 2 extra clocks.

Undecidability Result

- [TACAS 2004]
  The problem is 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.

An Overview of TIMES



Your Project

- You can use UPPAAL or Times, or both
  - to check for schedulability
  - correctness of your protocols/programs.
- You can play with the UPPAAL scheduler template.
- Problems:
  - Where do you get C? → Measurements.
  - Where do you get D? → Safety criteria.
  - Where do you get T? → Sampling, control algorithm…