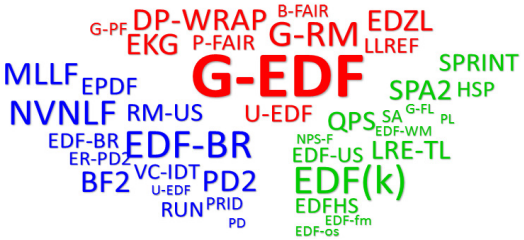
Formal approach for a verified implementation of Global EDF in Trampoline

Khaoula BOUKIR, Jean-Luc BÉCHENNEC, Anne-Marie DÉPLANCHE

Laboratoire des Sciences du Numériques de Nantes

October 10, 2018, Poitiers



Real-time scheduling algorithms

Implementation?

Real-time operating systems

- hardware platforms are basically modeled by "**m processors**"
- implementation constraints are completely ***abstracted***
- computational complexity overheads are ***neglected***

- scheduler programing at the kernel level is a ***difficult task***
- complexity of the physical target
- several implementation constraints (data structure management, interrupt handling, etc.)

our goal

Investigate the implementation of **global** multiprocessor scheduling policies within a **real platform**

our goal

Investigate the implementation of **global** multiprocessor scheduling policies within a **real platform**

To "formally" verify

- that the implementation of the scheduling policy is correct and always produces the expected behavior.
- functional properties of the implemented scheduler

why "formally" ?

- we reason on a model that describes exactly the behavior of the system + exploration of all possible cases (not only those given in simulation)

Our approach

1 - Implementing G-EDF

- use Trampoline [Béchennec et al.(2006)Béchennec, Briday, Faucou, and Trinquet] for implementation (scheduling policy : FTP + partitioned scheduling)
- adapt Trampoline to support global policies and implement G-EDF in it (As a start)

Our approach

1 - Implementing G-EDF

- use Trampoline [Béchennec et al.(2006)Béchennec, Briday, Faucou, and Trinquet] for implementation (scheduling policy : FTP + partitioned scheduling)
- adapt Trampoline to support global policies and implement G-EDF in it (As a start)

2 - Model elaboration

- model the implemented scheduler + Trampoline's components related to the scheduler (**based on the source code**)
- combine the elaborated models with the existing OS model

Our approach

1 - Implementing G-EDF

- use Trampoline [Béchennec et al.(2006)Béchennec, Briday, Faucou, and Trinquet] for implementation (scheduling policy : FTP + partitioned scheduling)
- adapt Trampoline to support global policies and implement G-EDF in it (As a start)

2 - Model elaboration

- model the implemented scheduler + Trampoline's components related to the scheduler (**based on the source code**)
- combine the elaborated models with the existing OS model

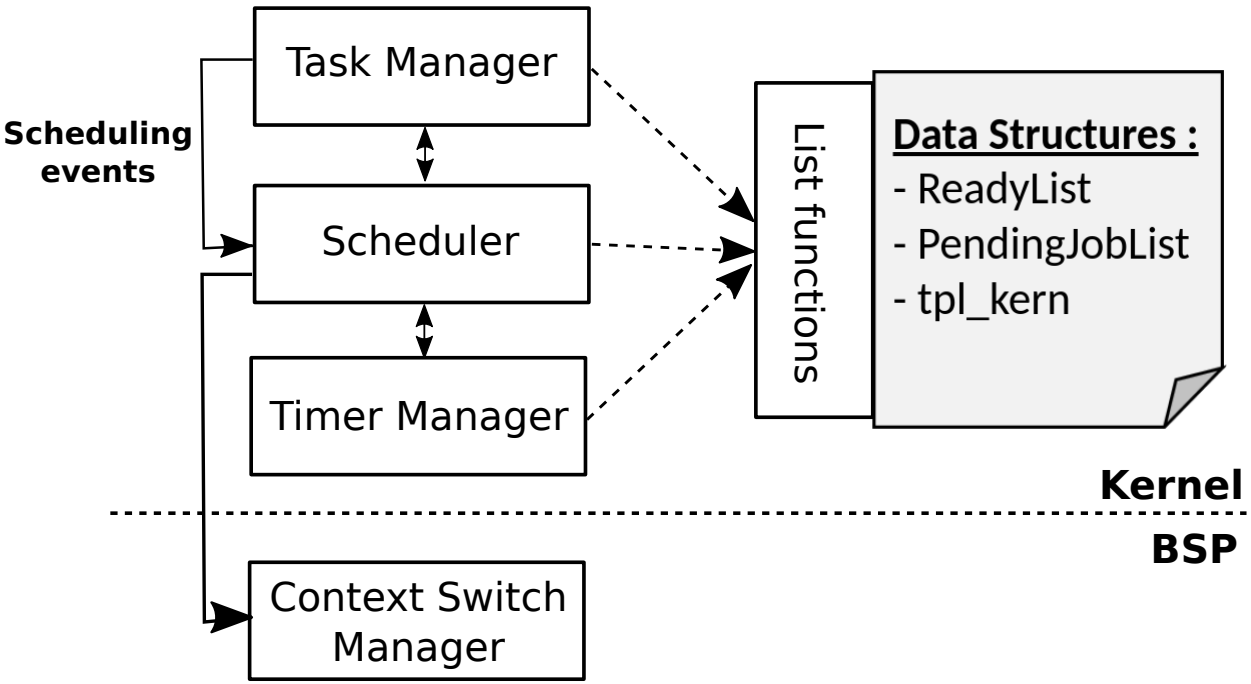
3 - Formal verification

- use model-checking to ensure that the implemented scheduler behaves correctly
- check reachability, safety and liveness properties

Plan

Identify scheduling related components

- the implementation of Trampoline's scheduler is ***kernel-based*** ==> determine the scope of the scheduler's action
- add/separate new components to Trampoline :
  - Timer manager : to handle the calculation/comparison of deadlines
  - list functions : gathers the functions used to manipulate task lists



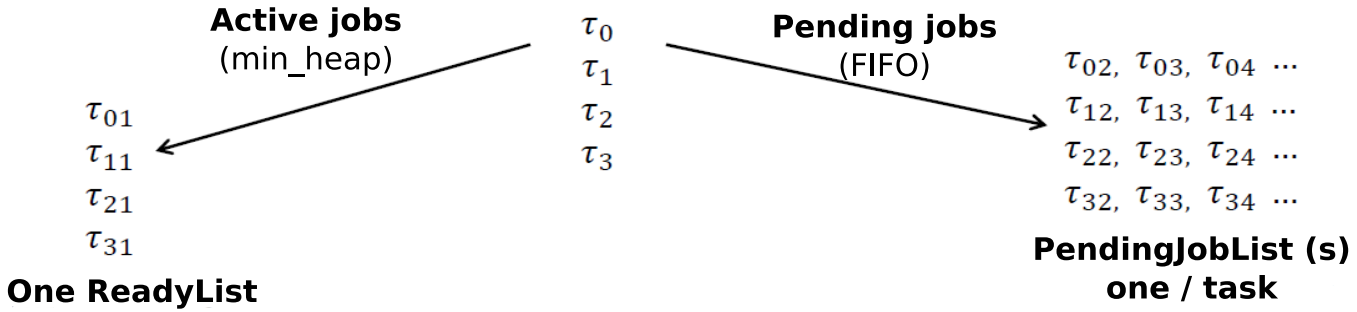
1- Time management :

- circular time representation : 32-bit time variable + 1 μs resolution
- `ActivateTask` ==> absolute deadline calculated and stored $d_i = current\_time + D_i$
- deadline comparison is performed using ICOTH algorithm [Carlini and Buttazzo(2003)]

1- Time management :

- circular time representation : 32-bit time variable + 1 μs resolution
- `ActivateTask` ==> absolute deadline calculated and stored $d_i = current\_time + D_i$
- deadline comparison is performed using ICOTH algorithm [Carlini and Buttazzo(2003)]

2- Task list management :



Existing model of Trampoline's kernel [Tigori et al.(2017)Tigori, Béchennec, Faucou, and Roux]

- synthesize the source code of the OS => elaboration of a complete model of Trampoline's kernel for single-core :

Extended Finite Automata + **UPPAAL Functions**

- variables used in the model are the control variables of the system.
- actions and conditions attached to each transition are the same ones of the source code of the system.

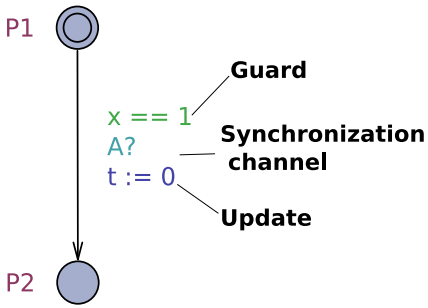
Existing model of Trampoline's kernel [Tigori et al.(2017)Tigori, Béchennec, Faucou, and Roux]

- synthesize the source code of the OS => elaboration of a complete model of Trampoline's kernel for single-core :

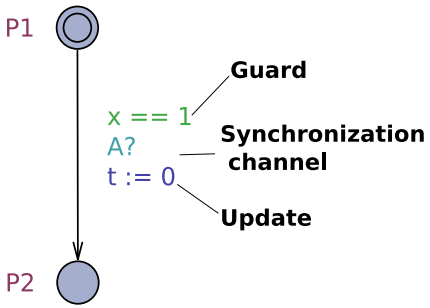
Extended Finite Automata + **UPPAAL Functions**

- variables used in the model are the control variables of the system.
- actions and conditions attached to each transition are the same ones of the source code of the system.

- retrieve the existing model
- follow the same logic
- add/modify the model for global scheduling



Actions

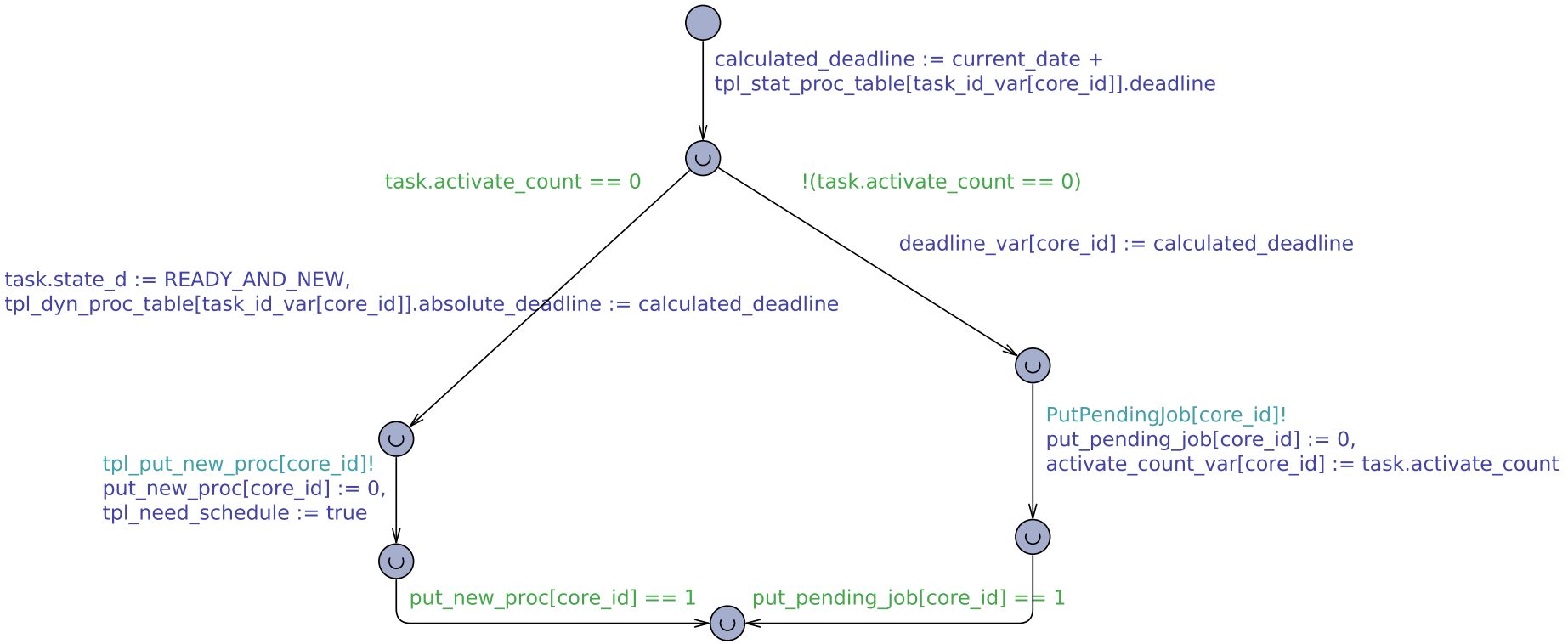


Actions

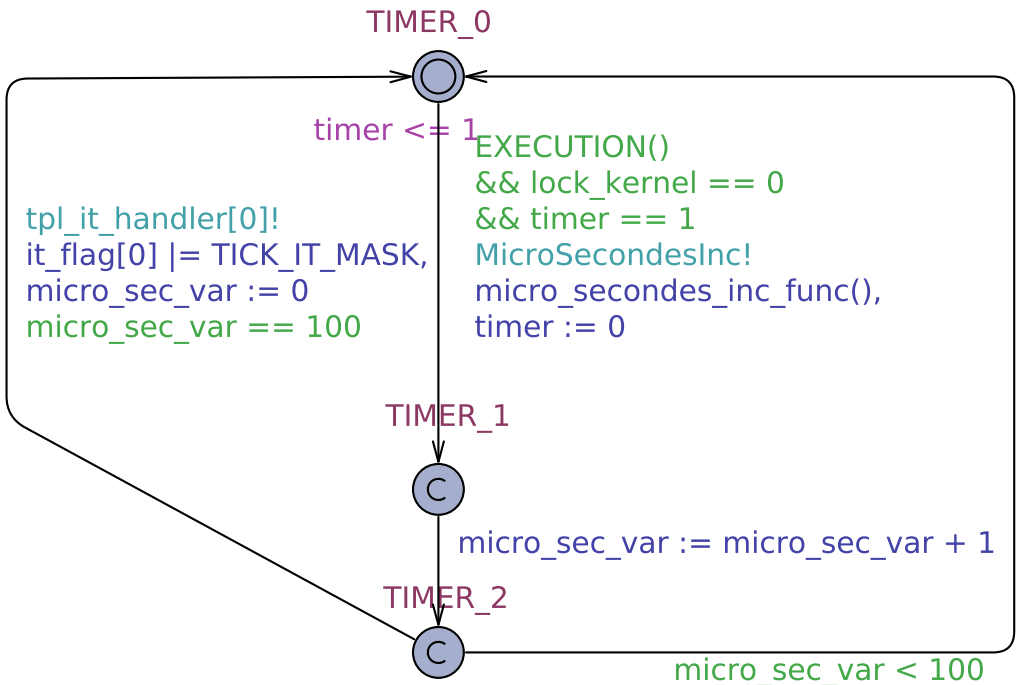
```
calculated_deadline = current_date + tpl_stat_proc_table[task_id].
deadline;

if(task.activate_count == 0){
 task.state_d = READY_AND_NEW;
 tpl_dyn_proc_table[task_id].absolute_deadline= calculated_deadline;
 tpl_put_new_proc(task_id);
 tpl_need_schedule = true;
}

else{
 PutPendingJob(calculated_deadline, task.activate_count);
}
```

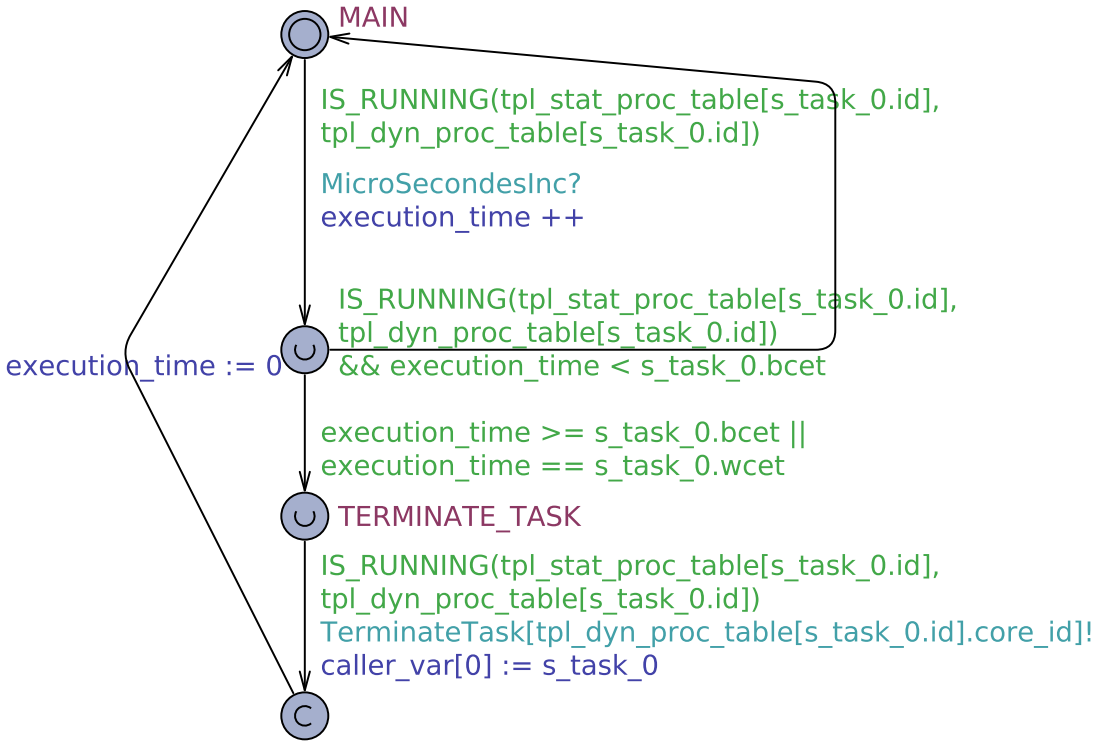


Timer model



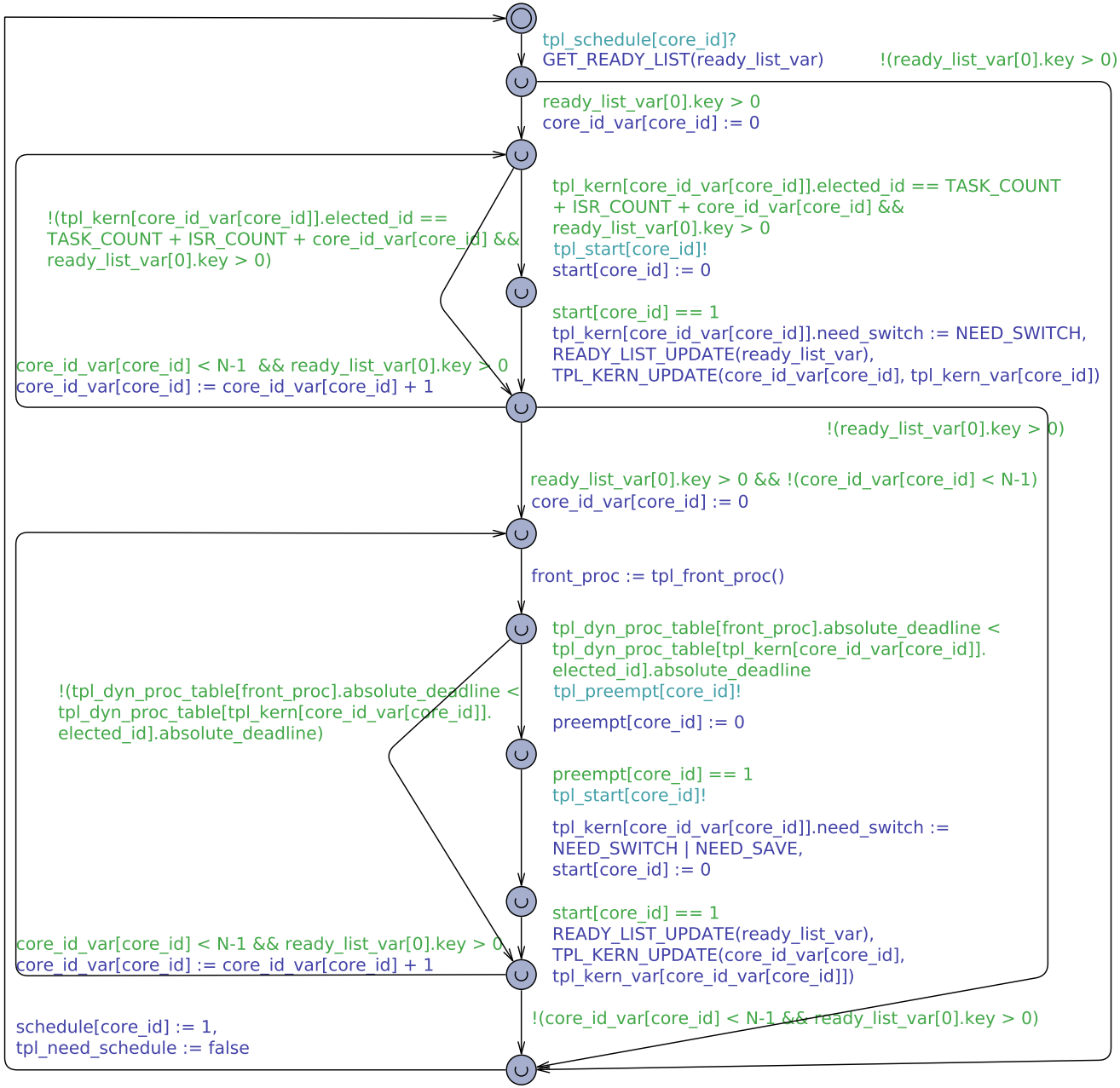
- time flows using a shared variable that represents the time in microseconds (as in the real implementation in Trampoline)
- the timer emits a call through a synchronization on a broadcast channel `MicroSecondesInc` to be executed.
- the OS is executed in zero time

Application model

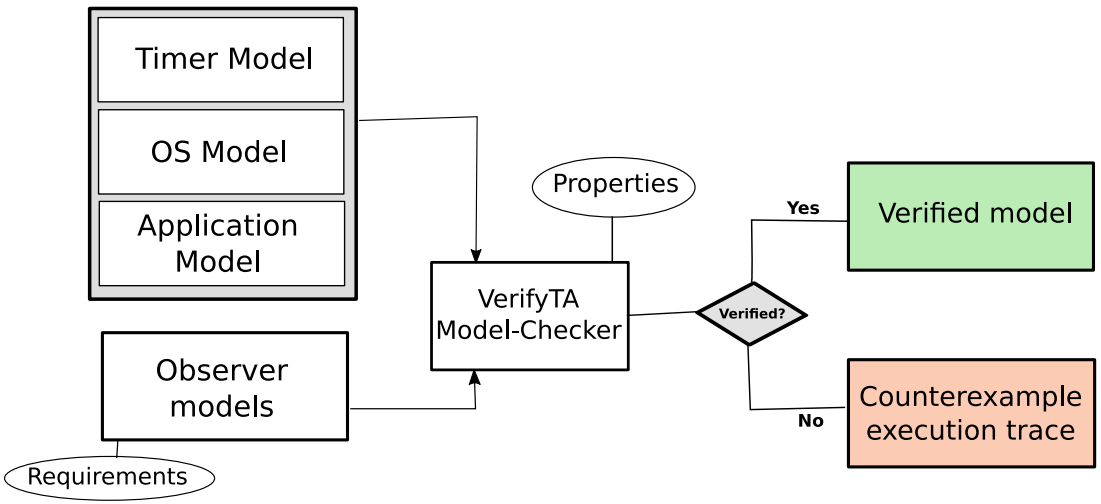


- the task is synchronized with the timer model
- the execution is constrained by the guard `IS_RUNNING()` which is true only if the task should be running. If the task is preempted or suspended, this guard returns false.
- the task can run for an execution time between its *bcet* and *wcet*

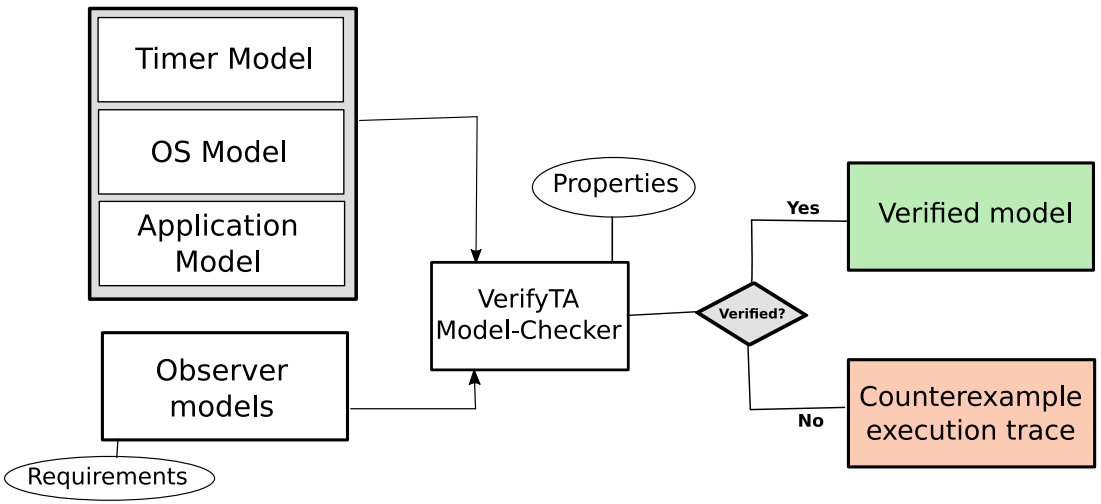
Scheduler's model

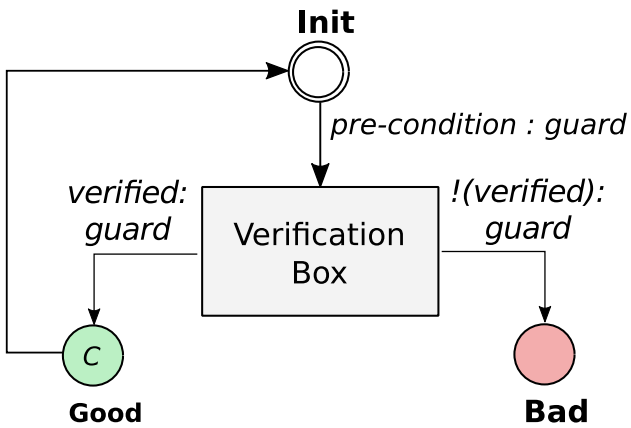


- verify the scheduler while it is calculating the scheduling decision
- combine the Timer model, the OS model and the application model
- insert observers that run in parallel with the complete model and observe its behavior.
- properties are tested on the complete model or the observers state using UPPAAL model-checker



- verify the scheduler while it is calculating the scheduling decision
- combine the Timer model, the OS model and the application model
- insert observers that run in parallel with the complete model and observe its behavior.
- properties are tested on the complete model or the observers state using UPPAAL model-checker





Reachability

It is used to verify that a `Good` observer states is reachable by using the path formula :
`E<> Observer.Good == true`

Liveness

"something good eventually happens"
used to check if there is a deadlock in the scheduler's model.

Safety

"nothing bad happens"
it is checked using the reachability property by verifying that a `Bad` observer state is never reached (`E<> Observer.Bad == false`)

Task configuration example (**first attempt**)

Testing the feasibility of our approach : The proposed application uses 20 periodic tasks to be executed on 2 cores, 10 with arbitrary and 10 with implicit deadlines.

✓ ***erified properties***

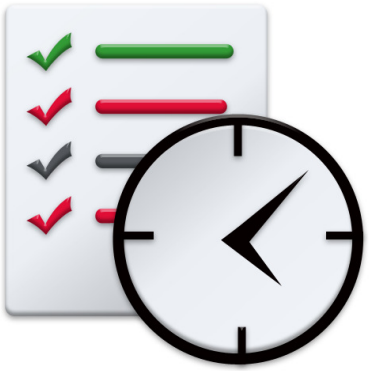
- activation/termination ==> call the scheduler
- `ReadyList` always sorted in an increasing deadline order
- two nodes of the ReadyList never have the same key

- deadlines are always calculated correctly
- deadline comparison is always performed correctly according to ICTOH algorithm

Task configuration example (**first attempt**)

Testing the feasibility of our approach : The proposed application uses 20 periodic tasks to be executed on 2 cores, 10 with arbitrary and 10 with implicit deadlines.

✓***erified properties***



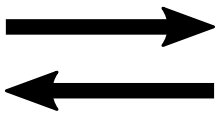
Scheduler

- a running task has always a lower deadline than any ready task
- an activated job always ends up running, in the `ReadyList` or in a `PendingJobList`
- there is no idle processor while the ReadyList is not empty
- the scheduler is deadlock free

Task configuration example (**first attempt**)

Testing the feasibility of our approach : The proposed application uses 20 periodic tasks to be executed on 2 cores, 10 with arbitrary and 10 with implicit deadlines.

✓ ***erified properties***



Context Switch Manager

- the context switch is performed whenever the scheduler indicates it

The whole system

- the system is deadlock free

Task configuration example

The proposed application uses 20 periodic tasks to be executed on 2 cores, 10 with arbitrary and 10 with implicit deadlines.

Runtime

Between **0.69s** and **99.04s**

Number of states

Between **12452** and **462358** state

Example of a detected bug

In fixed priority partitioned scheduling

A task is always assigned to the same processor which is specified statically

In dynamic global scheduling scheduling

Tasks can migrate between processors ==> the processor in which a task is executing is not always the same

Example of a detected bug

In fixed priority partitioned scheduling

A task is always assigned to the same processor which is specified statically

In dynamic global scheduling scheduling

Tasks can migrate between processors ==> the processor in which a task is executing is not always the same

Found bug

a statically assigned processor id was retrieved in a high-level function of the API layer

- **Consequences :** the context switching was performed after the scheduling according to this wrong id

In progress

- conduct reachability tests to all scheduler's states
- identify representative cases in order to specify an application model capable of covering all the cases of treatment of the scheduler
- cover a maximum number of situations the scheduler must deal with

Achieved

- implementation of G-EDF within Trampoline
- modeling the integrated scheduler and first results of its correctness
- the elaborated models are based on the source code of Trampoline
  ==> the presented properties does not conclude the complete verification of the scheduler, however it offers a global view on it

Perspectives and future works

- elaborate a complete application model for verification
- extend the same study to other sophisticated scheduling policies

Thank you for your attention

Bibliography

Jean-Luc Béchennec, Mikael Briday, Sébastien Faucou, and Yvon Trinquet.
Trampoline an open source implementation of the osek/vdx rtos specification.
In **Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on**, pages 62–69. IEEE, 2006.

Alessio Carlini and Giorgio C Buttazzo.
An efficient time representation for real-time embedded systems.
In **Proceedings of the 2003 ACM symposium on Applied computing**, pages 705–712. ACM, 2003.

Kabland Toussaint Gautier Tigori, Jean-Luc Béchennec, Sébastien Faucou, and Olivier Henri Roux.
Formal model-based synthesis of application-specific static rtos.
ACM Transactions on Embedded Computing Systems (TECS), 16(4) :97, 2017.