REVERT: Runtime Verification for Real-Time Systems

CISTER – Research Centre in Real-Time & Embedded Computing Systems

Sangeeth Kochanthara, Geoffrey Nelissen, David Pereira, Rahul Purandare
sangeeth1455@iiitd.ac.in, {grrpn,dmrpe}@isep.ipp.pt, purandare@iiitd.ac.in

Runtime Verification of Real-Time Systems

Limitations of classic (static) approaches:

- Number of reachable states too large for testing
- Potential blow-up when automatically exploring the system's state-space (e.g., model-checking)
- Limited automation in machine assisted proof construction tools (e.g., SMT solvers, proof-assistants)
- Difficulties in capturing data expected to be available only at run-time (need for abstraction leads to lack of precision)

Limitations of existing Runtime Verification solutions:

- Vast majority of tools developed for non-real-time applications;
- In most cases, it is difficult to capture extra-functional properties:
  - either no support at all; or
  - via complex specifications that are not accessible for the non-expert or the typical industrial practitioner
- Lack of a specification language that is user friendly, and that allows to capture distinct classes of timing properties

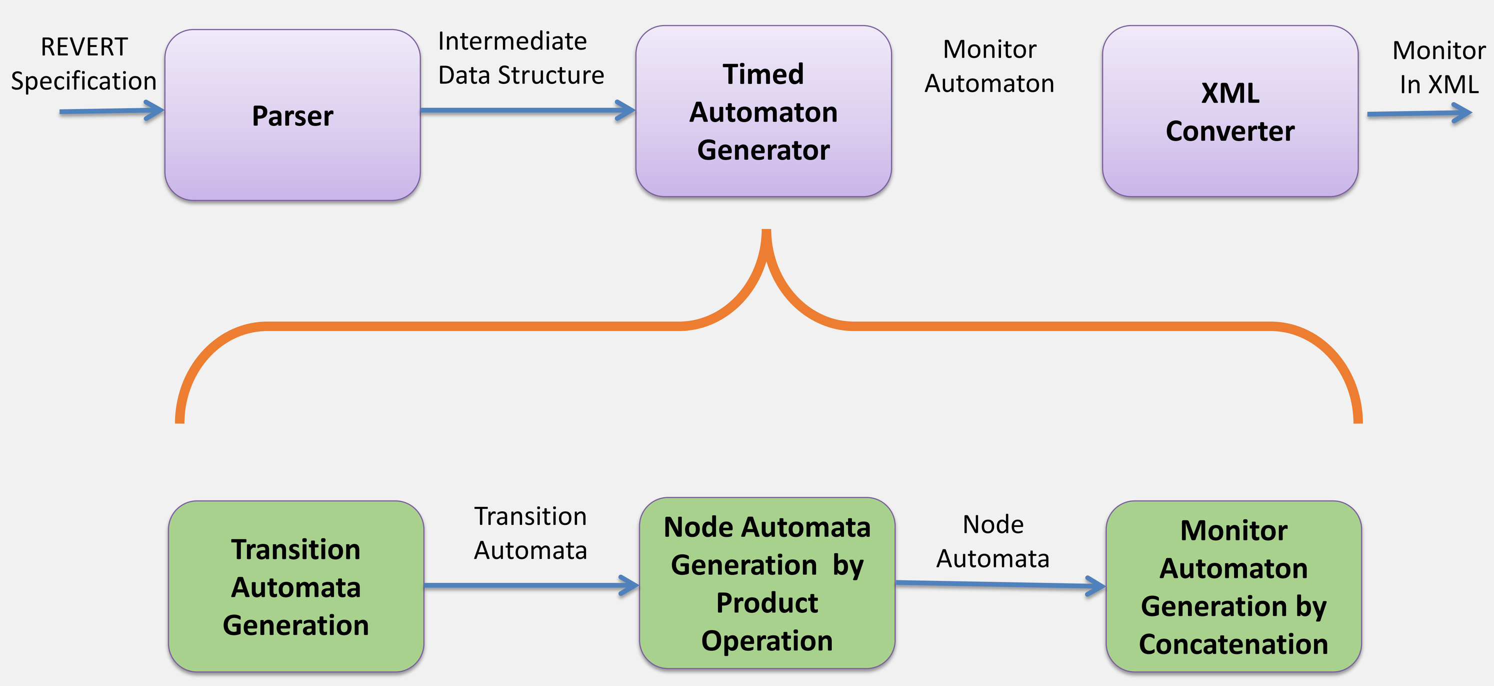
The REVERT Framework

1) A new specification language:

- Intuitive, easy to use domain specification language
- Capture changes in the system via guarded state-machine transitions between nodes (monitor states)
- Functional behavior as extended regular expressions
- Support for associating events with job specifications
- Three classes of timing constraints relevant for real-time systems: time, duration and jitter.
  - Timing constraint on sequences of events,
  - Execution time of a job,
  - Jitter on time and duration.
- Local variables and local code (e.g., for monitor initialization, calling counter-measure actions, etc)

2) A new monitor generation process:

1. REVERT specifications are parsed into intermediate data-structure;
2. Generation of the corresponding automata (via combination of intermediate types of finite automata)
3. Translation of the generated timed state-machine into XML format



Transition generation algorithm for timing constraints

every state $state_i$ is associated with two variables $REex_i$ and $REel_i$;
add start state ($state_0$) to the set $waiting\_states$;
reset clock variable $main\_clock$;
$REex_0 := \alpha$;
$REel_0 := 0$;
for *all* $state_i \in waiting\_states$ **do**
  for *all* $\xi \in \Sigma$ **do**
    if $\mathcal{D}_\xi(REex_i) \neq 0$ **then**
      if $\exists state_j \in waiting\_states$ *s. t.* $\mathcal{D}_\xi(REex_i) \in REex_j$ **then**
        $REel_j := REel_j \vee \mathcal{I}_\xi(REel_i)$;
      else if $\mathcal{D}_\xi(REex_i) = 1$ **then**
        create a new final state $state_j$;
        $REex_j := 1$;
        $REel_j := \mathcal{I}_\xi(REel_i)$;
      else
        add a new state $state_j$ to $waiting\_states$;
        $REex_j := \mathcal{D}_\xi(REex_i)$;
        $REel_j := \mathcal{I}_\xi(REel_i)$;
      end
      create a transition from $state_i$ to $state_j$;
    else
      $LSI :=$ longest suffix of $\mathcal{I}_\xi(REel_i)$ matched with $REel_j$ for any state $state_j \in waiting\_states$;
      if LSI *is empty* **then**
        create a transition from $state_j$ to $state_0$;
      else if *length of* $LSI = 1$ **then**
        create a self-loop on $state_i$ with $main\_clock$ reset;
      else
        add an auxiliary clock aux_clk_i;
        $RE_{pre} :=$ longest prefix of $\mathcal{I}_\xi(REel_i)$ before LSI;
        reset aux_clk_i at $state_k \in waiting\_states$ s. t. $RE_{pre} = REel_k$;
        create a transition from $state_i$ to $state_j$ with $main\_clock$ set to value of aux_clk_i;
      end
    end
  end
end

Transition automaton generation algorithm for time operator

Example Specification

```
use "T_Events.ev";
use "Ext_Procs.h";

monitor MyMon {

  observe { arrT, startT, suspT, blockedT,
        resumeT, unblockedT, complT }

  variables { failureReason : integer; }

  jobs {
    Job1 {
      start: {startT}
      suspend: {suspT, blockedT}
      resume: {resumeT, unblockedT}
      complete: {complT}
    }
  }

  nodes { NormalMode, RecoveryMode }
  initial { NormalMode }

  node NormalMode {
    init{
      resetAllSystemFlags();
    }
    constraints {
      c1: time(blockedT resumeT)) ≤ 2;
      c2: duration(Job1) ≤ 10;
    }
    transitions {
      fail_blocked_time: failure(c1) →
RecoveryMode {
        failureReason := 1;
        recover_from_blocking();
      }
      fail_duration: failure(c2) → RecoveryMode {
        failureReason := 2;
        recover_from_duration();
      }
    }
  }
  node RecoveryMode {
    init{
      initializeSystemRecovery();
    }
    constraints {
      c1[ERE]: _ complT;
    }
    transitions {
      job_completion: success(c1) →
NormalMode;
    }
  }
}
```

Concluding Remarks

- New specification language for runtime verification of RTSs
- Novel method to generate timed finite state machines that avoids state blowup in run-time
- Implemented the framework as a tool-chain

CISTER Research Unit (CEC/04234), co-financed by:
COMPETE 2020 · PORTUGAL 2020 · UNIÃO EUROPEIA Fundos Europeus Estruturais e de Investimento · FCT Fundação para a Ciência e a Tecnologia

CISTER Research Centre/INESC-TEC
ISEP, Polytechnic Institute of Porto
Rua Dr. Antº Bernardino de Almeida, 431
4249-015 Porto, Portugal

+351 228 340 502
www.cister.isep.ipp.pt
cister-info@isep.ipp.pt
facebook.com/cisterrealtime

isep Instituto Superior de Engenharia do Porto · INESCTEC