Runtime Verification for Real-Time Automotive Embedded Software

S. Cotard, S. Faucou, J.-L. Béchennec, A. Queudet, Y. Trinquet

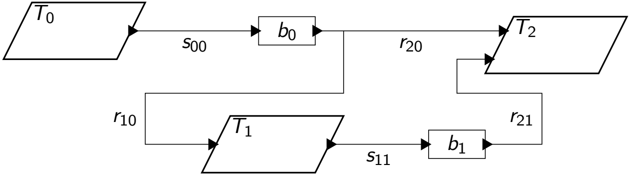
RENAULT

l'université nantes angers le mans

PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

10th school of Modelling and Verifying Parallel processes (MOVEP)

Motivating example

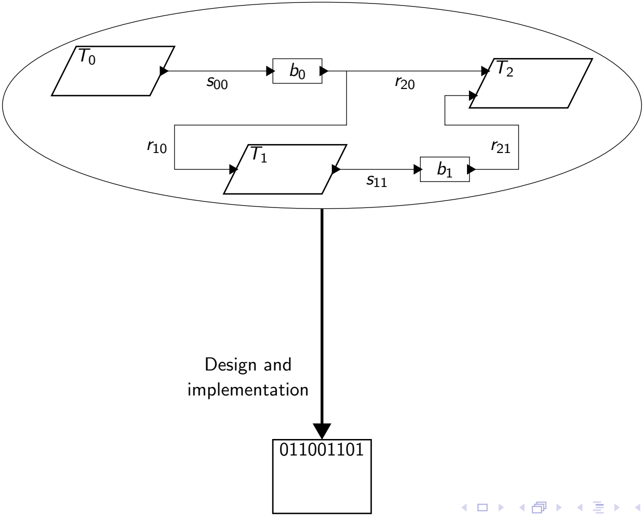


Safety constraint: T_2 requires the data from b_1, but also reads b_0 in order to perform a plausibility check. T_2 has to read the same instance of data.

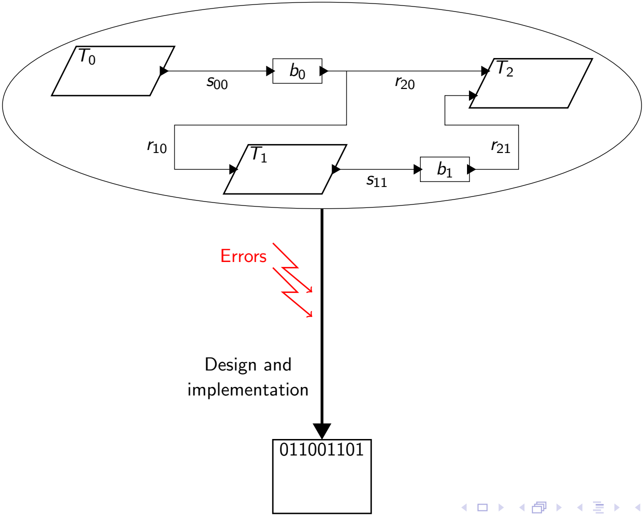
Requirement: consistency checking

Correctness property: when T_2 starts reading, the buffers are synchronized and stay synchronized until T_2 completes its execution

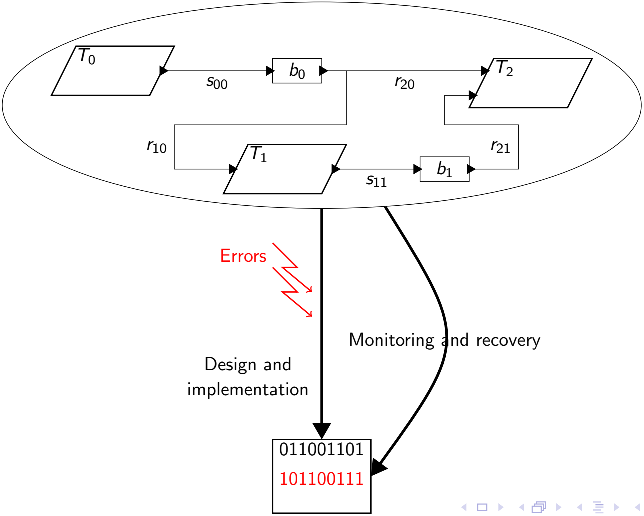
A possible solution: diversification



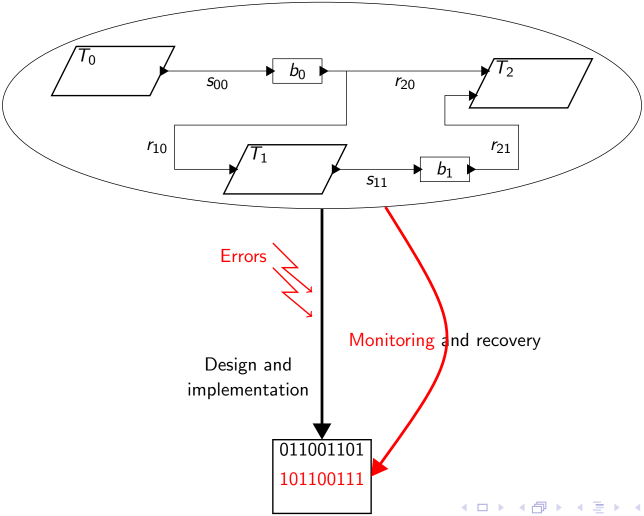
A possible solution: diversification



A possible solution: diversification



A possible solution: diversification



Context

Objectives

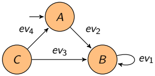
- based on formal methods
- compatible with functional and industrial constraints
  - small and deterministic detection latency
  - small and deterministic overheads (execution time, memory footprint)
  - compatible with multi-tiers system design process (the provided source code is not always modifiable)

Proposed solution

- runtime verification
- injection of monitors in the kernel

Formal methods

Model M of a system S



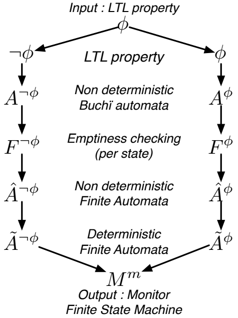
Property

ϕ

- Model checking: all runs of M satisfy ϕ ? (design time)
- Tests: some runs of S satisfy ϕ ? (design time)
- Runtime verification: does this run satisfy ϕ ? (online analysis)
  $\longrightarrow$ generate a monitor from M and ϕ that outputs a verdict in $\{\top, \bot, ?\}$

Our approach: runtime verification: step 1

[*Bauer et al, 2011*] solution



For ϕ and $\neg\phi$

1) Compute NBAs

2) Emptiness checking per state (derived F)

3) Compute NFAs using F

4) Compute DFAs

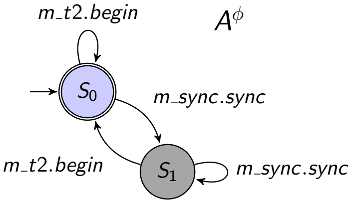
5) DFAs synchronization

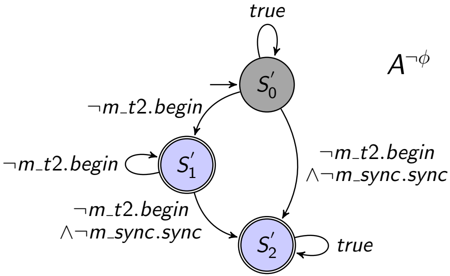
Our approach: runtime verification: step1

Property

$$\phi = \mathbf{G}\ ((m\_t2.firstb0 \vee m\_t2.firstb1) \implies (m\_sync.sync\ \mathbf{U}\ m\_t2.begin))$$

1) Computation of the NBAs



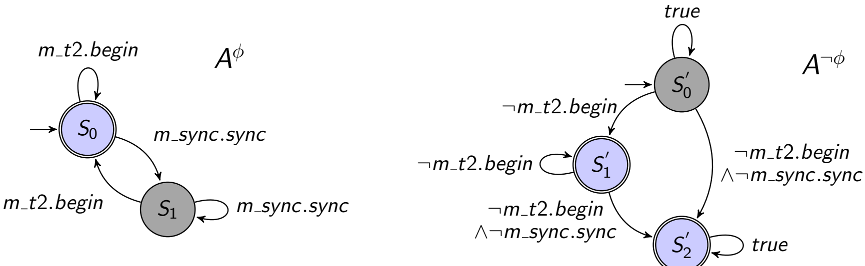


Our approach: runtime verification: step1

Property

$$\phi = \mathbf{G}\ ((m\_t2.firstb0 \vee m\_t2.firstb1) \implies (m\_sync.sync\ \mathbf{U}\ m\_t2.begin))$$

1) Computation of the NBAs



2) Emptiness checking per state

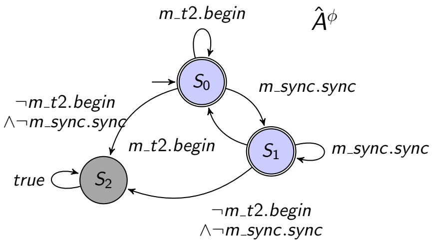
$$F^{\phi} = \{S_0, S_1\} \qquad F^{\neg\phi} = \{S_0', S_1', S_2'\}$$

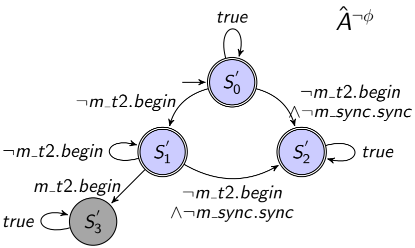
Our approach: runtime verification: step1

Property

$$\phi = \mathbf{G}\ ((m\_t2.firstb0 \vee m\_t2.firstb1) \implies (m\_sync.sync\ \mathbf{U}\ m\_t2.begin))$$

3) Computing NFAs using F and completes automata



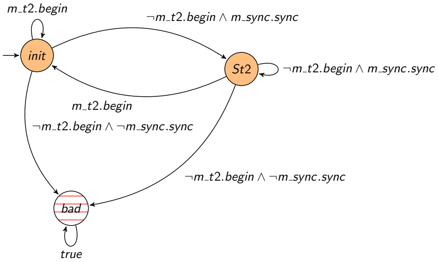


Our approach: runtime verification: step1

Property

$$\phi = \mathbf{G}\ ((m\_t2.firstb0 \vee m\_t2.firstb1) \implies (m\_sync.sync\ \mathbf{U}\ m\_t2.begin))$$

4) Determinization ⟶ Composition ⟶ Minimization



The intermediate monitor M^m reacts to changes in the values of the atomic propositions used in ϕ

Our approach: runtime verification: step1

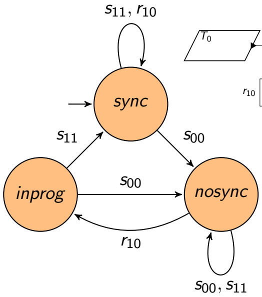
Intermediate monitor (M^m)

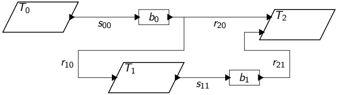
The intermediate monitor is the Moore machine given by $M^m = (Q^m, i^m, \rightarrow_m, \gamma^m)$ over 2^{AP}, the set of intercepted events

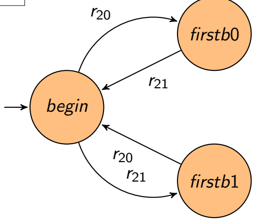
- Q^m is the finite set of states
- i^m is the initial state
- $\rightarrow_m \subset (Q^m \times 2^{AP}) \mapsto Q^m$ is the transition function
- $\gamma^m \subset Q^m \mapsto \mathbb{B}_3 = \{\top, \bot, ?\}$ is the output function

Our approach: runtime verification: step2

Input: system model + properties







m_sync: buffers synchronization

m_t2: *T*2 behavior

$$\mathbf{G}\ ((m\_t2.firstb0 \lor m\_t2.firstb1) \implies (m\_sync.sync\ \mathbf{U}\ m\_t2.begin))$$

Our approach: runtime verification: step2

Model of the system (A^s)

The model of the system is given by $A^s = (Q^s, i^s, \rightarrow_s)$ over Σ^s, the set of intercepted events

- Q^s is the finite set of states
- $i^s \in Q^s$ is the initial state
- $\rightarrow_s \subset (Q^s \times \Sigma^s) \mapsto Q^s$ is the transition function

We denote $\lambda^s \subset Q^s \mapsto 2^{AP}$, the labeling function that maps each state of the DFA to the set of atomic proposition true in this state.

Our approach: runtime verification: step2

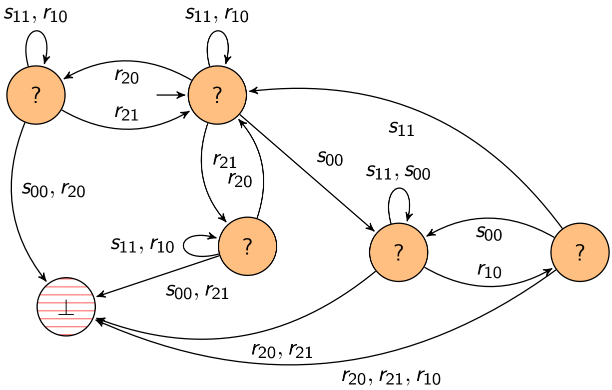
Final monitor computation (M')

The final monitor is defined by $M' = (Q', i', \rightarrow, \gamma')$ over Σ^s

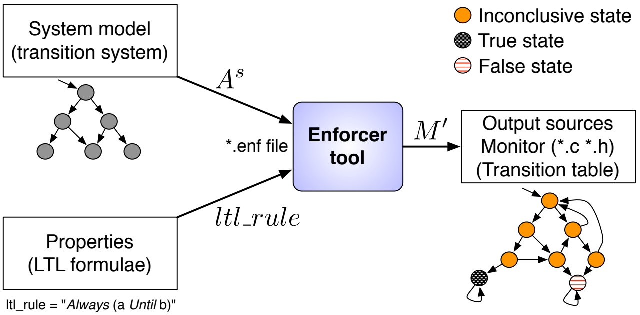
- $Q' = Q^s \times Q^m$
- $i' = (i^s, i^m)$
- $\rightarrow \subset (Q' \times \Sigma^s) \mapsto Q'$
  where $(q^s, q^m) \xrightarrow{\sigma} (r^s, r^m)$ iff $q^s \xrightarrow{\sigma}_s r^s$ and $q^m \xrightarrow{u}_m r^m$ and $u \subseteq \lambda^s(r^s)$ and $\gamma^m(q^m) =?$
- $\gamma' \subset Q' \mapsto \mathbb{B}_3$
  where $\gamma'(q^s, q^m) = \gamma^m(q^m)$

Our approach: runtime verification: step2

Output: a monitor



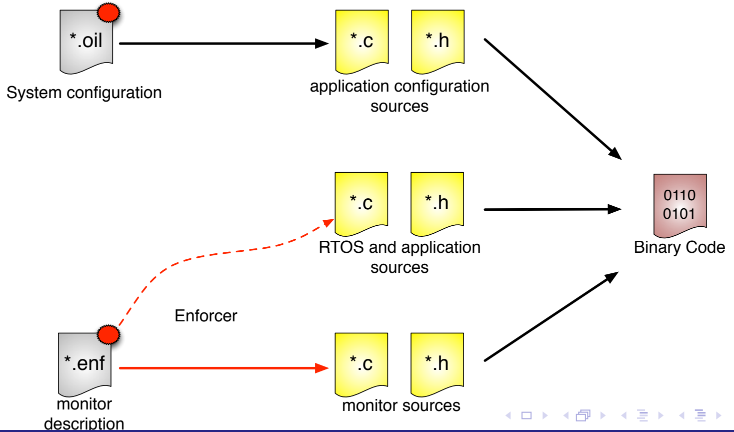
Enforcer: A tool for monitor synthesis



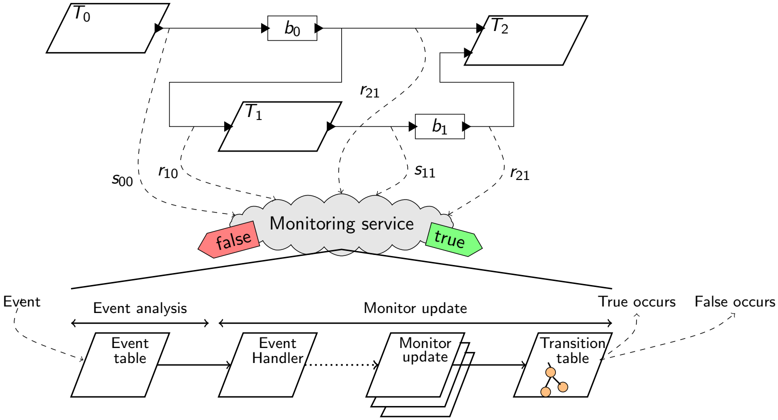
$$\underbrace{LTL \rightarrow NBA \rightarrow Intermediate\ Moore\ machine}_{step\ 1} \rightarrow \underbrace{Monitor}_{step\ 2}$$

Injection of the monitors in the kernel

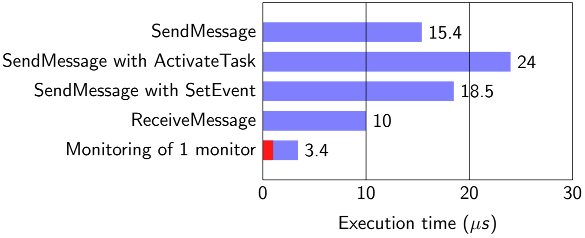
The Trampoline compilation chain (open-source implementation of AUTOSAR OS)



Architecture



Evaluation: computation overhead



- target running at 60 MHz
- composition of the overhead
  - $1\mu s$ to identify the event
  - $2.4\mu s$ to react per monitor interested in the event

Evaluation: memory footprint

Transition table	Monitor descriptor	Code size
ROM	RAM	ROM/RAM
30 *bytes* depends on the monitor 3 optimizations have been proposed	15 *bytes* constant per monitor	152 *bytes* (monitor update) constant 16 *bytes* (event handler) depends on the number of monitors per event

Conclusion

- approach has been implemented in a tool: Enforcer
  - freely available (see paper for URL)
- results show that runtime verification can be affordable for (static) industrial real-time embedded systems
  - kernel instrumention allows to achieve (guaranteed) low detection latency
  - static code and data generation allows to achieve low execution time overhead
  - system designer can pay time for memory
- future works
  - compute the theoretical bound on the size of the monitors (given the size of M and ϕ)
  - multicore extension (not only a matter of implementation)

Thank you for your attention