Assumption-Based Runtime Verification of Finite- and Infinite-State Systems

PhD thesis presentation

Chun Tian[1,2]

Advisor: Alessandro Cimatti[2]
Co-Advisor: Stefano Tonetta[2]

[1]University of Trento, Italy

[2]Fondazione Bruno Kessler, Italy

November 23, 2022

Outline

Formal Methods (FM)

Formal Methods (or Formal Verification) is the act of proving or disproving behavior correctness of systems (mathematical, physical or cyber-physical) with respect to certain formal specifications described in logical or mathematical formulas.

- Model Checking;
- Theorem Proving;
- Testing (e.g. Model-based[a]);
- Runtime Verification;

---

[a] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *LNCS 3472 - Model-Based Testing of Reactive Systems.*
Springer, Berlin, Heidelberg, 2005

Runtime Verification (RV)

- "The discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a run of a system under scrutiny (SUS) satisfies or violates a given correctness property"[a].
- "A dynamic analysis method aiming at checking whether a run of the system under scrutiny satisfies a given correctness property"[b].
- "A lightweight automatic formal verification technique for the dynamic analysis of systems, where a monitor observes executions produced by a system and analyzes its executions against a formal specification."

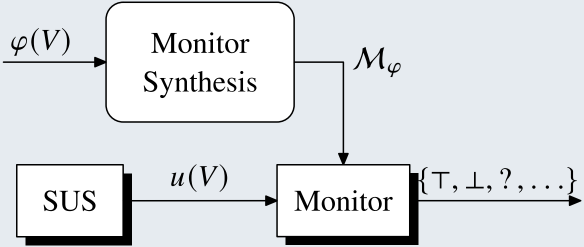
---

[a] M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009

[b] Y. Falcone, K. Havelund, and G. Reger. A tutorial on runtime verification. *Engineering Dependable Software Systems*, 34:141–175, 2013

Runtime Monitors - How They Work

Traditional RV approaches (before this thesis)



Monitor Synthesis (φ: property/specification, $\mathcal{M}_\varphi$: monitor, u: finite trace, $\top, \bot, \ldots$: verdicts)

$$\forall \varphi.\, \exists \mathcal{M}_\varphi.\, \forall u.\, \mathcal{M}_\varphi(u) = [\![u \models \varphi]\!] \in \{\top, \bot, ?\ \ldots\}$$

Runtime Monitors - How They Look Like

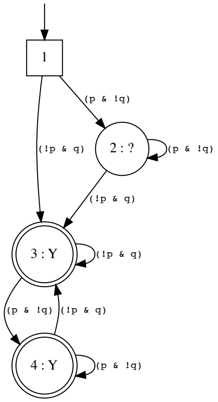
Monitoring specification: $\varphi = p \,\mathsf{U}\, q$ (p until q), *event-based*.

Samples

$$\mathcal{M}_{p\,\mathsf{U}\,q}(p) = ?$$
$$\mathcal{M}_{p\,\mathsf{U}\,q}(p \cdot p) = ?$$
$$\mathcal{M}_{p\,\mathsf{U}\,q}(p \cdot p \cdot q) = \top$$
$$\mathcal{M}_{p\,\mathsf{U}\,q}(p \cdot p \cdot q \cdot q) = \top$$

$$\mathcal{M}_{p\,\mathsf{U}\,q}(q) = \top$$
$$\mathcal{M}_{p\,\mathsf{U}\,q}(q \cdot p) = \top$$

NOTE: the trace state p corresponds to $p \wedge \neg q$ in the transition labels, etc.



Short History of the Present PhD Project (2017-2022)

1. Background: FBK's ES group already has a large code base of formal verification tools, covering SMT checking, LTL-based symbolic model checking, diagnosability and contract-based design, etc.
2. Initial work: implement "Runtime Verification for LTL and TLTL"[1] but in symbolic approach using BDDs (2018).
3. Assumptions, partial observability and resets (two RV 2019 papers[2][3]);
4. Re-implmenting finite-state algorithms using SMT and quantifier elimination;
5. Reductions between ABRV and Model Checking (BMC and IC3);
6. Incremental Bound Model Checking (RV 2021 paper[4]).

[1] A. Bauer, M. Leucker, and C. Schallhart. Runtime Verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4):14–64, Sept. 2011

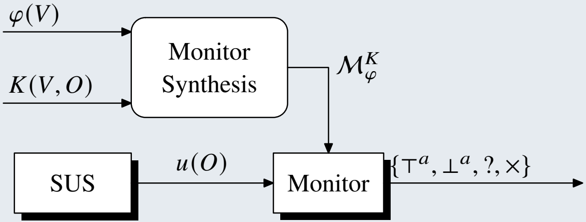
[2] A. Cimatti, C. Tian, and S. Tonetta. Assumption-Based Runtime Verification with Partial Observability and Resets. In *LNCS 11757 - Runtime Verification (RV 2019)*, pages 165–184. Springer, 2019

[3] A. Cimatti, C. Tian, and S. Tonetta. NuRV: A nuXmv Extension for Runtime Verification. In *LNCS 11757 - Runtime Verification (RV 2019)*, pages 382–392. Springer, 2019

[4] A. Cimatti, C. Tian, and S. Tonetta. Assumption-Based Runtime Verification of Infinite-State Systems. In *LNCS 12974 - Runtime Verification (RV 2021)*, pages 207–227. Springer International Publishing, Oct. 2021

Assumption-Based Runtime Verification (ABRV)

- Runtime monitors may be synthesized from a system model (assumptions);
- The ABRV-LTL semantics (as monitor verdicts) is based on LTL_3 semantics, adding one more verdict: $\times$ (*error* or *out-of-model*);
- Partially observable traces are naturally supported;
- The monitors can be reset, to evaluate the specification at later positions of the trace.



Resettable Monitors

- Traditionally the monitor only evaluates $[\![u \models \varphi]\!]$ $(= [\![u, 0 \models \varphi]\!])$;
- The resettable monitor takes as input some *reset* signals that change the reference time of evaluating monitor properties, e.g. from $[\![u, i \models \varphi]\!]$ to $[\![u, j \models \varphi]\!]$ $(j > i)$;
- The execution history of SUS is preserved during resets, possible impacts to monitoring outputs:
  1. Under assumptions, the belief states after resets may be different with initial belief states;
  2. With past operators, historical inputs may change the initial evaluation of a monitoring property.

Motivation of resettable monitors

1. Monotonic monitors: still meaningful after reaching conclusive verdicts;
2. Monitoring Past-Time LTL (to be explained).

Outline

Notations

Notations for finite-state words (traces)

Σ	finite alphabet (a set), e.g. $\{p, q\}$,
u, v, w	finite or infinite words (sequences of letters) over Σ,
ϵ	the empty word,
u_i	The (zero-indexed) ith letter of u,
u^i	the *sub-word* of w starting from u_i,
$\|u\|$	the *length* (i.e. number of letters) of u,
$u \cdot v$	the *concatenation* of a finite word u with another finite or infinite word v.

Additions for infinite-state words (traces)

V	a set of finite- or infinite-domain variables (as alphabet),
$\{x \mapsto 1, y \mapsto 2\}$	A (full or partial) value assignment of variables (in V),
$u = s_0 s_1 \dots s_n \cdots$	A finite or infinite sequence of value assignments.

Preliminaries

Satisfiability Modulo Theory (SMT)

First-order formulas are built as usual by proposition logic connectives, a given set of variables V and a first-order signature Σ, and are interpreted according to a given Σ-theory $\mathcal{T}$.
NOTE: In this thesis, we focus on $\mathcal{LRA}$ (linear arithmetic of reals).

(First-Order) Quantifier Elimination (QE)

QE methods convert first-order formulas into $\mathcal{T}$-equivalent quantifier-free formulas.
Formally speaking, if $\alpha(V_1 \cup V_2)$ is quantifier-free formula (of the theory $\mathcal{T}$) built by variables from the set $V_1 \cup V_2$, the role of quantifier elimination is to convert the first-order formula $\exists V_1.\alpha(V_1 \cup V_2)$ into an $\mathcal{T}$-equivalent formula $\beta(V_2)$, where β is quantifier-free and is built by only variables from V_2.

Fair Transition System (FTS) or Fair Kripke Structures (FKS)

$$K \doteq \langle V, \Theta, \rho, \mathcal{J} \rangle$$

where $V = \{x_1, \ldots, x_n\}$ is a finite set of variables, $\Theta(V)$ the *initial condition*, $\rho(V, V')$ the *transition relation*, and $\mathcal{J}$ a (finite) set of *justice conditions* as formulas over V.

Linear Temporal Logic

LTL Syntax: $\varphi ::= \mathrm{true} \mid \alpha \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi \mid \mathsf{Y}\,\varphi \mid \varphi\,\mathsf{S}\,\varphi$

where the (quantifier-free) formula α is built by a set of variables V and a first-order signature Σ, and is interpreted according to a Σ-theory $\mathcal{T}$. Other operators: $\mathsf{F}\,\varphi \doteq \mathrm{true}\,\mathsf{U}\,\varphi$ (*eventually*), $\mathsf{G}\,\varphi \doteq \neg\mathsf{F}\,\neg\varphi$ (*globally*), $\mathsf{O}\,\varphi \doteq \mathrm{true}\,\mathsf{S}\,\varphi$ (*once*), $\mathsf{H}\,\varphi \doteq \neg\mathsf{O}\,\neg\varphi$

Semantics ($w \in (2^V)^\omega$, $i \in \mathbb{N}$). $w, 0 \models \varphi$ is abbreviated as $w \models \varphi$.

$$
\begin{array}{ll}
w, i \models \mathrm{true} & \\
w, i \models \alpha & \text{iff } \alpha \in w_i \\
w, i \models \neg\varphi & \text{iff } w, i \not\models \varphi \\
w, i \models \varphi \vee \psi & \text{iff } w, i \models \varphi \text{ or } w, i \models \psi \\
w, i \models \mathsf{X}\,\varphi & \text{iff } w, i+1 \models \varphi \\
w, i \models \varphi\,\mathsf{U}\,\psi & \text{iff for some } k \geqslant i, w, k \models \psi \text{ and for all } i \leqslant j < k, w, j \models \varphi \\
w, i \models \mathsf{Y}\varphi & \text{iff } i > 0 \text{ and } w, i-1 \models \varphi \\
w, i \models \varphi\,\mathsf{S}\,\psi & \text{iff for some } k \leqslant i, w, k \models \psi \text{ and for all } k < j \leqslant i, w, j \models \varphi
\end{array}
$$

Translating LTL to ω-automata (1)

Elementary Variables (forming *principally temporal formulae*)

$$
\begin{aligned}
\mathrm{el}(\text{true}) &= \emptyset, & \mathrm{el}(\mathsf{X}\phi) &= \{\mathsf{x}_\phi\} \cup \mathrm{el}(\phi),\\
\mathrm{el}(p) &= \{p\}, & \mathrm{el}(\phi \mathsf{U} \psi) &= \{\mathsf{x}_{\phi\mathsf{U}\psi}\} \cup \mathrm{el}(\phi) \cup \mathrm{el}(\psi),\\
\mathrm{el}(\neg\phi) &= \mathrm{el}(\phi), & \mathrm{el}(\mathsf{Y}\phi) &= \{\mathsf{y}_\phi\} \cup \mathrm{el}(\phi),\\
\mathrm{el}(\phi \vee \psi) &= \mathrm{el}(\phi) \cup \mathrm{el}(\psi), & \mathrm{el}(\phi \mathsf{S} \psi) &= \{\mathsf{y}_{\phi\mathsf{S}\psi}\} \cup \mathrm{el}(\phi) \cup \mathrm{el}(\psi) \quad .
\end{aligned}
$$

Expansion Laws

$$
\psi \,\mathsf{U}\, \phi \Leftrightarrow \phi \vee (\psi \wedge \mathsf{X}(\psi \,\mathsf{U}\, \phi)), \qquad \psi \,\mathsf{S}\, \phi \Leftrightarrow \phi \vee (\psi \wedge \mathsf{Y}(\psi \,\mathsf{S}\, \phi)) \quad .
$$

NOTE: Expansion Laws do not hold for every LTL semantics over finite traces.

Translating LTL to Propositional Logic: $\chi(\cdot)$

$$
\chi(\psi) \doteq x_\psi \quad \text{for } \psi \text{ a principally temporal formula}
$$

$$
\chi(p \,\mathsf{U}\, q) = \chi(q \vee (p \wedge \mathsf{X}(p\mathsf{U}q))) = q \vee (p \wedge \mathsf{x}_{p\mathsf{U}q})
$$

Translating LTL to ω-automata (2)

Tableau construction[5]: $T_\varphi \doteq \langle V_\varphi, \Theta_\varphi, \rho_\varphi, \mathcal{J}_\varphi \rangle$, where

- The set of (Boolean) elementary variables: $V_\varphi \doteq \mathrm{el}(\varphi)$,
- Initial condition: (NOTE: $\chi(\varphi)$ to be used separately in the monitoring algorithm!)

$$\Theta_\varphi \doteq \chi(\varphi) \wedge \bigwedge_{\mathsf{y}_\psi \in \mathrm{el}(\varphi)} \neg \mathsf{y}_\psi,$$

- Transition relation:

$$\rho_\varphi \doteq \bigwedge_{\mathsf{x}_\psi \in \mathrm{el}(\varphi)} \left(\mathsf{x}_\psi \leftrightarrow \chi'(\psi)\right) \ \wedge \bigwedge_{\mathsf{y}_\psi \in \mathrm{el}(\varphi)} \left(\chi(\psi) \leftrightarrow \mathsf{y}'_\psi\right),$$

- Justice set (a fairness condition):

$$\mathcal{J}_\varphi \doteq \left\{\chi(\psi \, \mathsf{U} \, \phi) \rightarrow \chi(\phi) \mid \mathsf{x}_{\psi \mathsf{U} \phi} \in \mathrm{el}(\varphi)\right\} \ .$$

- Fair states:

$$\mathcal{F}^K_\varphi \doteq \{s \mid T_\varphi, s \models \mathsf{E} \bigwedge_{\psi \in \mathcal{J}_\varphi} \mathsf{GF}\psi\} \ .$$

[5] Y. Kesten, A. Pnueli, and L.-o. Raviv. Algorithmic Verification of Linear Temporal Logic Specifications. In *LNCS 1443 - Automata, Langugues and Programming (ICALP 1998)*, pages 1–16. Springer, Berlin, Heidelberg, May 1998

Outline

Recall: LTL_3 semantics

- Three-valued semantics of LTL formula φ over a finite word $u \in \Sigma^*$:

$$[\![u, i \models \varphi]\!]_3 = \begin{cases} \top, & \text{if } \forall w \in \Sigma^\omega.\ u \cdot w, i \models \varphi, \\ \bot, & \text{if } \forall w \in \Sigma^\omega.\ u \cdot w, i \not\models \varphi, \\ ?, & \text{otherwise } . \end{cases}$$

  with $[\![u \models \varphi]\!]_3$ denoting $[\![u, 0 \models \varphi]\!]_3$.
- $[\![u \models \varphi]\!]_3 = \top/\bot$ if all extensions of u satisfy/violate φ;
- Monitor definition:

$$\mathcal{M}_\varphi(u) = [\![u \models \varphi]\!]_3 \ .$$

ABRV definitions

Set of fair paths

$$\mathcal{L}^K(u) \doteq \left\{\sigma \in \mathcal{L}(K) \;\middle|\; \forall i < |u|.\, \sigma_i \models_{\mathcal{T}} u_i\right\}$$

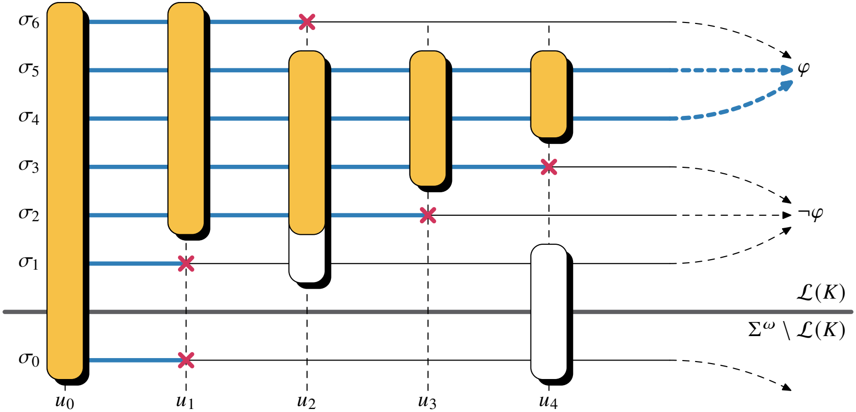
ABRV-LTL semantics

$$[\![u, i \models \varphi]\!]_4^K \doteq \begin{cases} \times, & \text{if } \mathcal{L}^K(u) = \emptyset \\ \top^{\mathrm{a}}, & \text{if } \mathcal{L}^K(u) \neq \emptyset \text{ and } \forall w \in \mathcal{L}^K(u).\ w, i \models \varphi \\ \bot^{\mathrm{a}}, & \text{if } \mathcal{L}^K(u) \neq \emptyset \text{ and } \forall w \in \mathcal{L}^K(u).\ w, i \models \neg\varphi \\ ? & \text{otherwise} \ . \end{cases}$$

ABRV monitor

$$\mathcal{M}_\varphi^K(u, i) \doteq [\![u, i \models \varphi]\!]_4^K \quad .$$

ABRV illustration



In this case, $\mathcal{M}_\varphi^K(u_0 u_1 u_2 u_3 u_4) = \top^{\mathrm{a}}$.

Reductions between ABRV and Model Checking

Model Checking to ABRV

MC problem $K \models \varphi$ can be reduced to ABRV monitoring on *empty traces* using K as assumptions.

$$\mathcal{M}_{\varphi}^{K}(\epsilon) = \begin{cases} \top^{\mathrm{a}}, & \text{if } [\![K \models \varphi]\!] = \top \text{ (and } [\![K \models \neg\varphi]\!] = \bot\text{)}, \\ \bot^{\mathrm{a}}, & \text{if } [\![K \models \varphi]\!] = \bot \text{ (and } [\![K \models \neg\varphi]\!] = \top\text{)}, \\ ?\,, & \text{if } [\![K \models \varphi]\!] = [\![K \models \neg\varphi]\!] = \bot \text{ (counterexamples exist on both sides)}, \\ \times, & \text{if } [\![K \models \varphi]\!] = [\![K \models \neg\varphi]\!] = \top \text{ (i.e. } \mathcal{L}(K) = \emptyset\text{, i.e. } K \text{ is an empty model).} \end{cases}$$

ABRV to Model Checking (Corollary: ABRV of Infinite-State Systems is Undecidable)

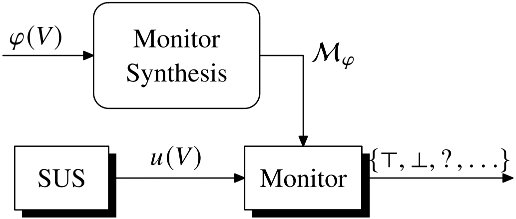
The ABRV monitoring problem $\mathcal{M}_{\varphi}^{K}(u)$ can be reduced to two MC calls $[\![K \times S_u \models \varphi]\!]$ and $[\![K \times S_u \models \neg\varphi]\!]$, where S_u represents all traces compatible with u.

$[\![K \times S_u \models \varphi]\!]$	$[\![K \times S_u \models \neg\varphi]\!]$	$\mathcal{M}_{\varphi}^{K}(u)$
$\top$	$\top$	$\times$
$\top$	$\bot$	$\top^{\mathrm{a}}$
$\bot$	$\top$	$\bot^{\mathrm{a}}$
$\bot$	$\bot$	$?$

Outline

Monitoring LTL$_3$[6]

1. An LTL property φ;
2. Two symbolic automata T_φ and $T_{\neg\varphi}$;
3. A finite input trace $u \in \Sigma^*$;
4. Two belief states $T_\varphi(u)$ and $T_{\neg\varphi}(u)$;



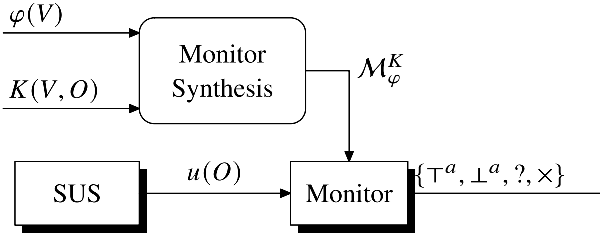
The monitor output is given by:

$T_\varphi(u)$	$T_{\neg\varphi}(u)$	$\mathcal{M}_\varphi(u)$
not $\emptyset$	not $\emptyset$	? (inconclusive)
not $\emptyset$	$\emptyset$	$\top$ (conclusively true)
$\emptyset$	not $\emptyset$	$\bot$ (conclusively false)
$\emptyset$	$\emptyset$	(impossible)

[6] A. Bauer, M. Leucker, and C. Schallhart. Runtime Verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4):14–64, Sept. 2011

Monitoring ABRV-LTL

1. An LTL property φ;
2. A model K used as the assumption;
3. Two symbolic automata $K \otimes T_\varphi$ and $K \otimes T_{\neg\varphi}$;
4. A finite input trace $u \in \Sigma^*$;
5. Two belief states $(K \otimes T_\varphi)(u)$ and $(K \otimes T_{\neg\varphi})(u)$;



The monitor output is given by:

$(K \otimes T_\varphi)(u)$	$(K \otimes T_{\neg\varphi})(u)$	$\mathcal{M}_\varphi^K(u)$
not $\emptyset$	not $\emptyset$	? (inconclusive)
not $\emptyset$	$\emptyset$	$\top^a$ (true under assumption)
$\emptyset$	not $\emptyset$	$\bot^a$ (false under assumption)
$\emptyset$	$\emptyset$	$\times$ (out of model)

The Finite-State Symbolic Monitor

1. $T_\varphi \doteq \langle V_\varphi, \Theta_\varphi, \rho_\varphi, \mathcal{J}_\varphi \rangle := \texttt{ltl\_translation}(\varphi)$;
2. $T_{\neg\varphi} \doteq \langle V_\varphi, \Theta_{\neg\varphi}, \rho_\varphi, \mathcal{J}_\varphi \rangle := \texttt{ltl\_translation}(\neg\varphi)$;
3. $V := V_K \cup V_\varphi$;
4. $\mathcal{F}_\varphi^K := \texttt{fair\_states}(K \otimes T_\varphi)$, $\mathcal{F}_{\neg\varphi}^K := \texttt{fair\_states}(K \otimes T_{\neg\varphi})$;
5. $\langle r_\varphi, r_{\neg\varphi} \rangle := \langle \Theta_K \wedge \Theta_\varphi \wedge \mathcal{F}_\varphi^K, \Theta_K \wedge \Theta_{\neg\varphi} \wedge \mathcal{F}_{\neg\varphi}^K \rangle$;
6. If $|u| > 0$, $\langle r_\varphi, r_{\neg\varphi} \rangle := \langle r_\varphi \wedge \mathsf{obs}(u_0), r_{\neg\varphi} \wedge \mathsf{obs}(u_0) \rangle$;
7. Loop for $1 \leqslant i < |u|$:
   - 7.1 If $\mathsf{res}(u_i) = \bot$:
     $r_\varphi := \mathrm{fwd}(r_\varphi, \rho_K \wedge \rho_\varphi)(V) \wedge \mathsf{obs}(u_i) \wedge \mathcal{F}_\varphi^K$;
     $r_{\neg\varphi} := \mathrm{fwd}(r_{\neg\varphi}, \rho_K \wedge \rho_\varphi)(V) \wedge \mathsf{obs}(u_i) \wedge \mathcal{F}_{\neg\varphi}^K$;
   - 7.2 If $\mathsf{res}(u_i) = \top$ (i.e. monitor is reset):
     $r := r_\varphi \vee r_{\neg\varphi}$;
     $r_\varphi := \mathrm{fwd}(r, \rho_K \wedge \rho_\varphi)(V) \wedge \chi(\varphi) \wedge \mathsf{obs}(u_i) \wedge \mathcal{F}_\varphi^K$;
     $r_{\neg\varphi} := \mathrm{fwd}(r, \rho_K \wedge \rho_\varphi)(V) \wedge \chi(\neg\varphi) \wedge \mathsf{obs}(u_i) \wedge \mathcal{F}_{\neg\varphi}^K$;
8. $b_1 := (r_\varphi = \mathrm{false})$ and $b_2 := (r_{\neg\varphi} = \mathrm{false})$;
9. Output 4 verdicts according to logical values of b_1 and b_2.

Outline

Quantifier Elimination for Computing Belief States

(For an incremental monitoring algorithm ...)

Definition (forward image)

The *forward image* of a set of states $\psi(V)$ on $\rho(V, V')$ is given by

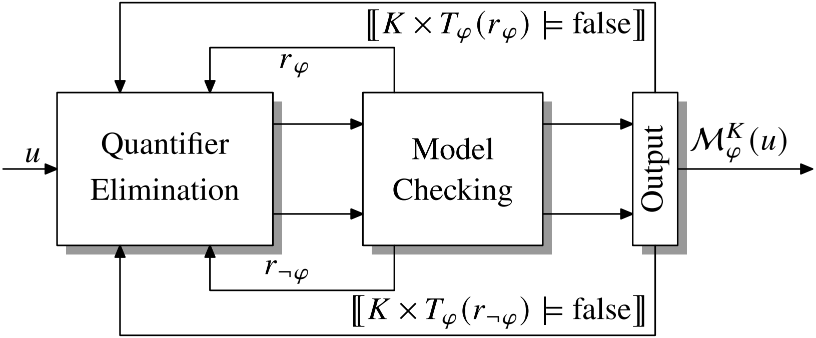
$$\mathrm{fwd}(\psi(V), \rho(V, V'))(V) \doteq \big(\exists V.\ \rho(V, V') \wedge \psi(V)\big)[V/V']$$

where $[V/V']$ denotes the substitution of (free) variables in V' with the corresponding one in V.

The need of QE procedures

First-order quantifier elimination (QE) can be involved to convert a forward image into an equivalent quantifier-free formula that can be directly sent to SMT solvers, etc.

ABRV reduced to MC and QE (1)



$\neg[\![K \times T_\varphi(r_\varphi) \models \text{false}]\!]$	$\neg[\![K \times T_\varphi(r_{\neg\varphi}) \models \text{false}]\!]$	$\mathcal{M}_\varphi^K(\cdot)$
$\top$	$\top$	?
$\top$	$\bot$	$\top$[a]
$\bot$	$\top$	$\bot$[a]
$\bot$	$\bot$	$\times$

The Infinite-State Symbolic Monitor

1. $T_\varphi \doteq \langle V_\varphi, \Theta_\varphi, \rho_\varphi, \mathcal{J}_\varphi \rangle :=$ `ltl_translation`(φ);
2. $T_{\neg\varphi} \doteq \langle V_\varphi, \Theta_{\neg\varphi}, \rho_\varphi, \mathcal{J}_\varphi \rangle :=$ `ltl_translation`$(\neg\varphi)$;
3. $V := V_K \cup V_\varphi$;
4. $\mathcal{F}^K_\varphi :=$ `fair_states`$(K \otimes T_\varphi)$, $\mathcal{F}^K_{\neg\varphi} :=$ `fair_states`$(K \otimes T_{\neg\varphi})$;
5. $\langle r_\varphi, r_{\neg\varphi} \rangle := \langle \Theta_K \wedge \Theta_\varphi \wedge \mathcal{F}^K_\varphi, \Theta_K \wedge \Theta_{\neg\varphi} \wedge \mathcal{F}^K_{\neg\varphi} \rangle$;
6. If $|u| > 0$, $\langle r_\varphi, r_{\neg\varphi} \rangle := \langle r_\varphi \wedge \mathrm{obs}(u_0), r_{\neg\varphi} \wedge \mathrm{obs}(u_0) \rangle$;
7. Loop for $1 \leqslant i < |u|$:
   - 7.1 If $\mathrm{res}(u_i) = \bot$:
     $r_\varphi :=$ `quantifier_elimination`$(V, \rho_K \wedge \rho_\varphi \wedge r_\varphi) \wedge \mathrm{obs}(u_i)$;
     $r_{\neg\varphi} :=$ `quantifier_elimination`$(V, \rho_K \wedge \rho_\varphi \wedge r_{\neg\varphi}) \wedge \mathrm{obs}(u_i)$;
   - 7.2 If $\mathrm{res}(u_i) = \top$ (i.e. monitor is reset): ...
8. $b_1 := \neg$`model_checking`$(\langle V, r_\varphi, \rho_K \wedge \rho_\varphi, \mathcal{J}_K \cup \mathcal{J}_\varphi \rangle, \text{false})$;
9. $b_2 := \neg$`model_checking`$(\langle V, r_{\neg\varphi}, \rho_K \wedge \rho_\varphi, \mathcal{J}_K \cup \mathcal{J}_\varphi \rangle, \text{false})$;
10. Output 4 verdicts according to logical values of b_1 and b_2.

Basic Optimizations

Basic optimization ideas

- o1 If the monitor has already reached conclusive verdicts ($\top^a$ or $\bot^a$), then for the runtime verification of the next input state *at most one* MC call is need.
- o2 Before calling model checkers to detect the emptiness of a belief state (w.r.t. fairness), an SMT checking can be done first, to check if the belief state formula can be satisfied or not.
- o3 When `monitor2` is used as online monitor, the same LTL properties are sent to LTL model checkers with different models and are internally translated into equivalent FTS.
- o4 Call the faster but incomplete BMC (or any other MC procedure which only detects counter-examples) before calling a unbounded model checker such as ic3ia.

Theorem

Assuming `BMC` *always find the counter-example whenever it exists,* `IC3_IA` *is called at most twice in the "online" version of* `monitor2` *with all above optimizations.*

Incremental BMC: The Idea (1)

Basic observation

All models used for model checking in (non)emptiness checking $[\![K \times T_\varphi(r_\varphi) \models \mathrm{false}]\!]$ only differ at the initial condition.

BMC encoding of belief states

The belief states after a sequence of observations $u_0 u_1 \cdots u_n$, denoted by $\mathrm{bs}(u_0 u_1 \cdots u_n)$, can be inductively given by

$$\mathrm{bs}(u_0)(V) = I(V) \wedge u_0(V),$$
$$\mathrm{bs}(u_0 u_1 \cdots u_{i+1})(V) = \mathrm{fwd}\big(\mathrm{bs}(u_0 u_1 \cdots u_i)(V), T(V, V')\big)(V) \wedge u_{i+1}(V)$$

Theorem (Equisatisfiability)

When $k > 1$, the SMT formulas $I(V_0) \wedge u_0(V_0) \wedge \bigwedge_{j=0}^{k-1} \left[T(V_j, V_{j+1}) \wedge u_{j+1}(V_{j+1})\right]$ and $\mathrm{bs}(u_0 u_1 \cdots u_k)(V)$ are equi-satisfiable.

Incremental BMC: The Idea (2)

Bounded Model Checking

Given a FSM M and an LTL specification f, the idea is to look for counter-examples of maximum length k, and to generate a Boolean formula which is satisfiable if and only if such counter-example exists. The checking of $M \models_k \mathrm{E}f$ is equivalent to the satisfiability problem of a Boolean formula $[[M, f]]_k$ defined as follows:

$$[[M, f]]_k := [[M]]_k \wedge [[f]]_k = I(V_0) \wedge \left(\bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \right) \wedge [[f]]_k$$

where $I(V_0)$ and $T(V_i, V_{i+1})$ represent the initial condition and transition relation of M, respectively, unrolled with state variables from certain discrete time.

A question (the answer is NO)

If a previous BMC process (on M, f) has completed at certain value of k, say $k = k_0$ (assuming $k_0 > 0$), with a counter-example found, and then the model M is *updated* to M' by having one or more *observations* at certain time, does the new BMC process for M', f need to restart from $k = 0$?

Incremental BMC: The Idea (3)

Definition (Step constraints (of models))

A *step constraint* (aka *observation*) of the model

$$M \doteq \langle V, I(V), T(V, V'), \mathcal{J} \rangle$$

is a pair $\langle t, s(V) \rangle$, where t is an integer indicating discrete time, and $s(V)$ is a formula of V. The model M updated with $\langle t, s(V) \rangle$, denoted by $M + \langle t, s(V) \rangle$, is a new model M' defined below:

$$\begin{aligned} M' \doteq \langle & V \cup \{c\}, \\ & I(V), \\ & T(V, V') \wedge (c' = \min\{c+1, t+1\}) \wedge ((c = t) \rightarrow s(V)), \\ & \mathcal{J} \rangle \end{aligned}$$

where $c \notin V$ is a fresh variable used as a counter, whose finite domain is from 0 to $t+1$.

Incremental BMC: The Idea (4)

1. Suppose there's just one observation $M' = M + \langle t, s(V) \rangle$, and the previous BMC process terminates at k_0.
2. A counter-example was found when doing SMT checking of $[[M, f]]_{k_0}$ (*satisfiable* for k_0, and for all $k < k_0$, $[[M, f]]_k$ is *unsatisfiable*).
3. Consider the form of $[[M', f]]_k$, where $M' = M + \langle t, s(V) \rangle$ and $k \leqslant k_0$. There are two possible cases:
   3.1 If $t \leqslant k_0$, then $[[M']]_k$ and $[[M]]_k \wedge s(V_t)$ are equi-satisfiable, where $s(V_t)$ occurs either in the initial condition ($t = 0$) or the transition relation ($t > 0$) of M',
   3.2 If $t > k_0$, then $[[M', f]]_k$ is unsatisfiable due to the domain of counter variable c.

Lemma (Incremental BMC)

The BMC process for M', f, if done incrementally from the BMC process of M, f, should start from $[[M', f]]_{k_0}$ (the accumulated formula left by previous BMC process) but may need to keep unrolling until reaching $[[M', f]]_{\max(k_0, t)}$, which contains the current observation $\langle t, s(V) \rangle$.

Outline

ptLTL - LTL without future temporal operators

When there's no future temporal operators, there's only one state in the belief states holding only the present state and the history. (Effective rewriting-based approach exists for monitoring ptLTL.)

Definition (alternative ptLTL semantics $[\![u \models_{\mathrm{p}'} \varphi]\!]$)

Let $u = s_0 \cdots s_{n-1}$ (thus $|u| = n$)[a],

$$
\begin{array}{lll}
u \models_{\mathrm{p}'} \mathrm{true} & & \\
u \models_{\mathrm{p}'} p & \text{iff } p \in s_{n-1} & \\
u \models_{\mathrm{p}'} \neg\varphi & \text{iff } u \not\models_{\mathrm{p}'} \varphi & \\
u \models_{\mathrm{p}'} \varphi \vee \psi & \text{iff } u \models_{\mathrm{p}'} \varphi \text{ or } u \models_{\mathrm{p}'} \psi & \\
u \models_{\mathrm{p}'} \mathsf{Y}\varphi & \text{iff } n > 1 \text{ and } u|_{n-1} \models_{\mathrm{p}'} \varphi & (\text{was: } u|_{n-1} \models_{\mathrm{p}'} \varphi \text{ if } n > 1 \text{ or } u \models_{\mathrm{p}'} \varphi \text{ if } n = 1) \\
u \models_{\mathrm{p}'} \varphi \,\mathsf{S}\, \psi & \text{iff for some } j \leqslant n, u|_j \models_{\mathrm{p}'} \psi \text{ and for all } j < i \leqslant n, u|_i \models_{\mathrm{p}'} \varphi &
\end{array}
$$

[a] K. Havelund and D. A. Peled. Runtime Verification: From Propositional to First-Order Temporal Logic. In *LNCS 11237 - Runtime Verification (RV 2018)*, pages 90–112. Springer, Cham, Oct. 2018

ptLTL and LTL_3

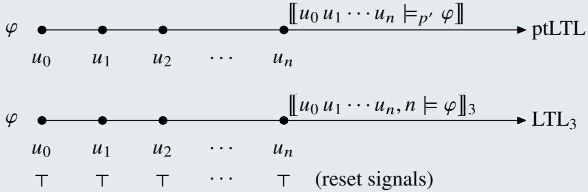
Lemma

The LTL_3 semantics of any ptLTL formula φ with respect to any non-empty finite trace u (thus $|u| > 0$) is always conclusive: $\forall i < |u|.\ [\![u, i \models \varphi]\!]_3 = \top$ *or* $\bot$.

Theorem

The alternative semantics of any ptLTL formula φ with respect to any non-empty finite trace u can be expressed by LTL_3 semantics, i.e., $[\![u \models_{\mathrm{p}'} \varphi]\!] = [\![u, |u| - 1 \models \varphi]\!]_3$.

Monitoring ptLTL (alternative semantics) by LTL_3 monitor with resets

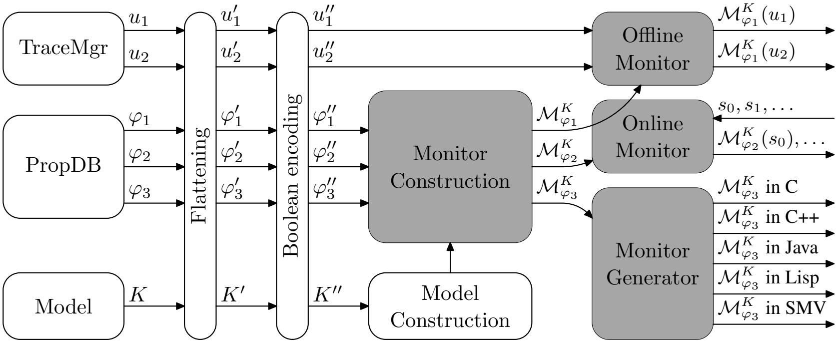


Outline

Classification of NuRV according to the Taxonomy

Concepts	Branches	Classification of NuRV
Specification	data	*propositional*
	output	*verdict*, *stream*
	time (logical)	*total order (linear time)*
	time (physical)	$\mathbb{N}$ *(discrete time)*
	modality	*all (future, past, current)*
	paradigm	*all (declarative, operational)*
Monitor	decision procedure	*automata-based*
	generation	*all (implicit, explicit)*
	execution	*all (interpreted, direct)*
Deployment	stage	*all (online, offline)*
	synchronisation	*synchronous*
	architecture	*centralised*
	placement	*all (inline, outline)*
	instrumentation	*none*
Reaction	active	none
	passive	*specification output*
Trace	information	*all (events, states)*
	sampling	*all (event-triggered, time-triggered)*
	evaluation	*points*
	precision	*all (precise, imprecise)*
	model	*infinite (LTL), finite* (LTL_3, *ptLTL*)

NuRV: The low-level architecture



- nuXmv components: TraceMgr, PropDB, Boolean encoding, Model, Flattening, Model Construction;
- NuRV components: Monitor Construction, Offline Monitor, Online Monitor, Monitor Generator.

Other features of NuRV

Existing features

- New code generation languages after RV 2019: Python and Prolog (beside C, C++, Java, Common Lisp and SMV).
- LLVM IR code generation (platform independent, no runtime library);
- CORBA-based monitor server (IDL proposed for next CRVs);
- Code generation with high-level API using scalar variables;
- "Symbolic monitors" calling NuRV itself as a dynamic library;

Planned features (optimizations)

- Multi-property monitoring;
- Finite-state monitoring using SAT solver (based on MiniSAT)
- Full SMV language support in code generation (e.g. SMV arrays).

Outline

Model Checking Monitors Generated in SMV

nuXmv to check the following properties:

- The *rough* correctness (w/o resets):
  $(\mathsf{F}\ \texttt{M.\_true}) \to \varphi$ or $(\mathsf{F}\ \texttt{M.\_false}) \to \neg\varphi$;
- The monotonicity of monitors:
  $\mathsf{G}\ \texttt{M.\_unknown} \vee (\texttt{M.\_unknown}\ \mathsf{U}\ \texttt{M.\_concl})$;
- Comparison of two monitors (M1: with assumptons, M2: w/o assumptions):
  $\neg\mathsf{F}\ \texttt{Av}$, where $\texttt{Av} := (\texttt{M1.\_concl} \wedge \neg\,\texttt{M2.\_concl})$.
- The correctness of resets:
  $\mathsf{X}^n\,(\texttt{M.\_reset} \wedge \mathsf{X}\ (\neg\,\texttt{M.\_reset}\ \mathsf{U}\ \texttt{M.\_true})) \to \mathsf{X}^n\varphi$.

Observation

The full correctness of LTL monitors cannot be model checked by LTL itself. (Epistemic operator needed)

Tests on LTL patterns

Mattew Dwyer's LTL patterns (55 in total)

- Collected from over 500 specifications from at least 35 different sources.[a]
- 11 groups: Absence, Existence, Bounded Existence, Universality, Precedence, Response (2-causes-1, 1-cause-2), Precedence Chain (2-stimulus-1, 1-stimulus-2), Response Chain, Constrained Chain;
- 5 scopes: Globally, Before, After, Between, After-Until.

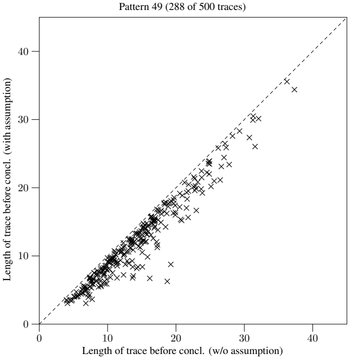
---

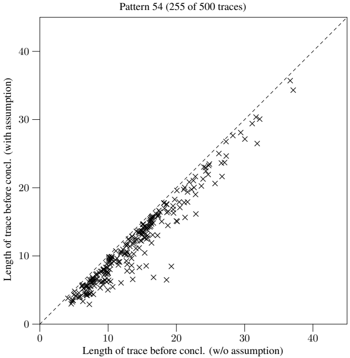
[a] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specifications for Finite-State Verification. In *Proceedings of the 21st International Conference on Software Engineering*, pages 411–420, New York, USA, 1999. ACM Press

All patterns are synthesized into working level 1 monitors (with or w/o assumptions); 500 random traces (each with 50 states) were used to compare the monitoring results.

Tests - The Value of Assumption

- Assumption: transitions to s-state occur at most 2 times: $((\neg s)\,\mathsf{W}\,(s\,\mathsf{W}\,((\neg s)\,\mathsf{W}\,(s\,\mathsf{W}\,(\mathsf{G}\,\neg s)))))$
  $(\varphi\,\mathsf{W}\,\psi \doteq (\mathsf{G}\,\varphi) \vee (\varphi\,\mathsf{U}\,\psi))$
- Pattern 29: s responds to p after q until r: $\mathsf{G}\,(q \wedge \neg r \rightarrow ((p \rightarrow (\neg r\,\mathsf{U}\,(s \wedge \neg r)))\,\mathsf{W}\,r))$.
- Pattern 49: s,t responds to p after q until r:
  $\mathsf{G}\,(q \rightarrow (p \rightarrow (\neg r\,\mathsf{U}\,(s \wedge \neg r \wedge \mathsf{X}\,(\neg r\,\mathsf{U}\,t))))\,\mathsf{U}\,(r \vee \mathsf{G}\,(p \rightarrow (s \wedge \mathsf{X}\,\mathsf{F}\,t))))$





Performance Tests on Dwyer's LTL patterns (1)

Pattern 49

$\varphi = \mathsf{G}\,(q \rightarrow (p \rightarrow (\neg r\,\mathsf{U}\,(s \wedge \neg r \wedge \mathsf{X}\,(\neg r\,\mathsf{U}\,t))))\,\mathsf{U}\,(r \vee \mathsf{G}\,(p \rightarrow (s \wedge \mathsf{X}\,\mathsf{F}\,t))))$
("s, t responds to p after q until r"),
with $q := (0 \leqslant i)$, $r := (0.0 \leqslant x)$, $i \in [-500, 500]$ and $x \in [-0.500, 0.500]$.

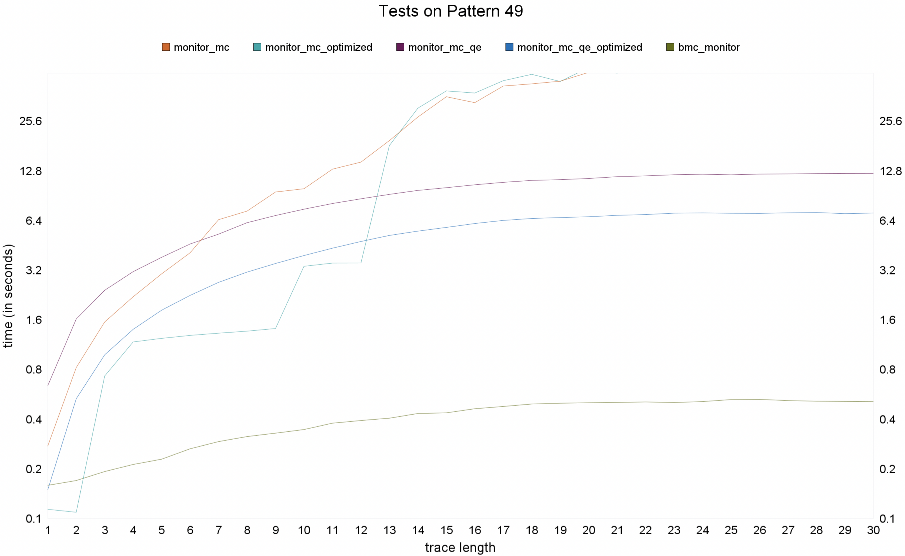
RV assumptions

"The p-transition (i.e., from $\neg p$ to p) happens at most 4 times"

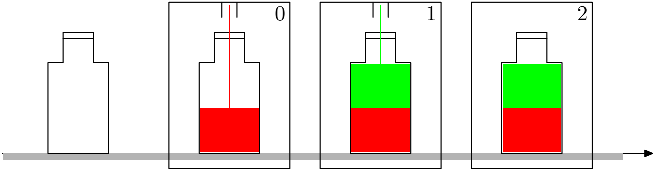
Other settings

The length of input traces increases from 1 to 30.

Performance Tests on Dwyer's LTL patterns (2)



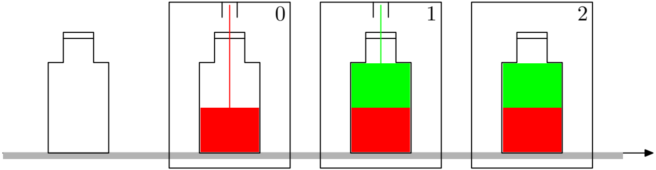
Tests on a Factory Model (1)



Model variables:

- bottle_present[0-2]: there exists a bottle at position 0–2;
- bottle_ingr1[0-2]: red ingredient in the bottle at position 0–2;
- bottle_ingr2[0-2]: green ingredient in the bottle at position 0–2;
- move_belt: the belt is moving;
- new_bottle: new bottle at position 0 before the belt starts to move.

Tests on a Factory Model (2)



- Whenever the belt is not moving and there is a bottle at position 2, both ingredients are filled in that bottle:

$$\varphi = \mathsf{G}\left((\texttt{bottle\_present[2]} \wedge \neg\,\texttt{move\_belt}) \rightarrow (\texttt{bottle\_ingr1[2]} \wedge \texttt{bottle\_ingr2[2]})\right)$$

- Two monitors: `M1` (with assumptions), `M2` (w/o assumptions).
- Model checking spec: $\neg\mathsf{F}\ \texttt{Av}$, where $\texttt{Av} := (\texttt{M1.\_concl} \wedge \neg\,\texttt{M2.\_concl})$.
- *Conclusion*: the monitor `M1` is predictive, it outputs $\bot^{\mathrm{a}}$ soon after the fault at position 0.

Conclusions (and Thank You)

Contributions

- ABRV – an extended RV framework with assumptions, partial observability and resets;
- Symbolic monitoring algorithms for finite- and infinite-state systems;
- The feature-rich NuRV tool implementation.

Key technical discoveries

1. Resetting ABRV monitors by taking the union of belief states;
2. Incremental BMC (for efficiently outputting inconclusive verdicts);

Future directions

- More expressive specification languages with quantifiers;
- Dense-time (hybrid) logics;
- Out-of-order trace inputs;
- Probabilistic models as assumptions.

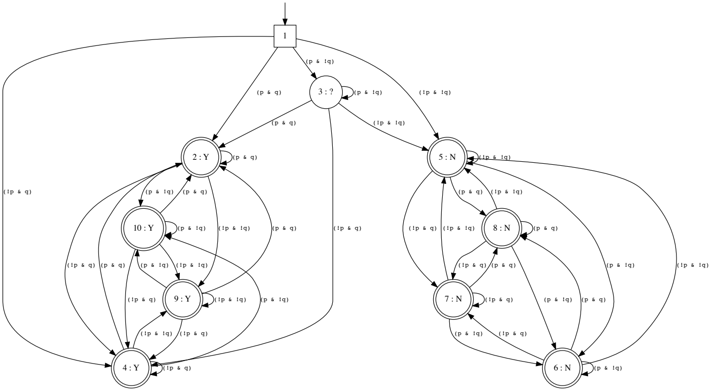
Runtime Monitors - How They Look Like (2)

Monitoring specification: $\varphi = p \,\mathsf{U}\, q$ (p until q), *state-based*.

Samples

$$\mathcal{M}_{p\,\mathsf{U}\,q}(\{p\}) = ?$$
$$\mathcal{M}_{p\,\mathsf{U}\,q}(\{p\} \cdot \{p\}) = ?$$
$$\mathcal{M}_{p\,\mathsf{U}\,q}(\{p\} \cdot \{p\} \cdot \{q\}) = \top$$
$$\mathcal{M}_{p\,\mathsf{U}\,q}(\{p\} \cdot \{p\} \cdot \{q\} \cdot \{q\}) = \top$$
$$\mathcal{M}_{p\,\mathsf{U}\,q}(\{q\}) = \top$$
$$\mathcal{M}_{p\,\mathsf{U}\,q}(\emptyset) = \bot$$

NOTE: $\{p\}$ means $p \wedge \neg q$; $\{p, q\}$ means $p \wedge q$; $\emptyset$ means $\neg p \wedge \neg q$, etc.
NOTE2: monitors are *monotonic*.

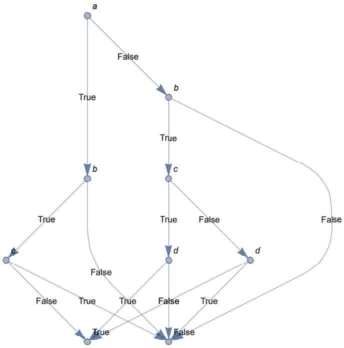


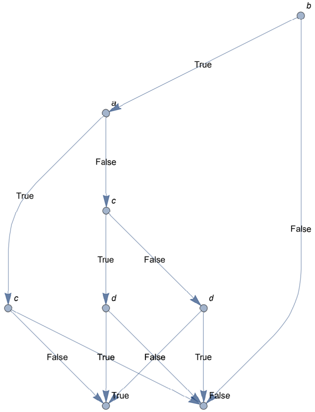
Brief History of Runtime Verification (2001-2019)

1. Domination of (linear) temporal logic variants (LTL, MTL, QTL, MFOTL, ...)
2. Monitor-Oriented Programming (MOP) framework (2003,2007): JavaMOP and BusMOP;
3. Efficient algorithms using BDDs (Klaus Havelund and Doron Peled; 2005-2018);
4. Model-based Runtime Verification Framework (Zhao et al.; 2009);
5. Comparisons of LTL Semantics for Runtime Verification (Bauer, Leucker and Christian; 2010);
6. Runtime Verification for LTL and TLTL (Bauer, Leucker and Christian; 2011);
7. International Competition on Runtime Verification (CRV) (2014, 2015, 2016);
8. A Taxonomy of RV tools (Falcone et al.; 2012, 2018).
9. Beyond Runtime Verification: Runtime Enforcement, Runtime Adaptation, Decentralized RV, ...

Binary Decision Diagrams (BDD)

Binary decision diagrams (BDD) provide a data structure for representing and manipulating Boolean functions in symbolic form. (NuRV benefits from the canonical representations of Boolean formulae using BDDs, when synthesizing explicit-state monitors, using CUDD 2.4)
Example: $(a \wedge b \wedge \neg c) \vee (\neg a \wedge b \wedge c \wedge d) \vee (b \wedge \neg c \wedge \neg d)$, before and after reordering:





Benefits of the Symbolic Approach

- Symbolic LTL translation has $O(n)$-complexity.[7]
- BDD is faster than explicit-state automata constructions.
- Partial Observability is supported in computing $T_\varphi(u)$ or $(K \otimes T_\varphi)(u)$:

$$\forall p \in AP.\ p \text{ is non-observable } \Rightarrow p \text{ can be any value.}$$

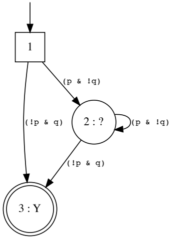
- The monitor can be easily *reset* by taking $(K \otimes T_\varphi)(u) \cup (K \otimes T_{\neg\varphi})(u)$ as the new initial belief states. Roughly speaking, future predictions are neutralized, resting the history of the current input trace. (A key contribution)

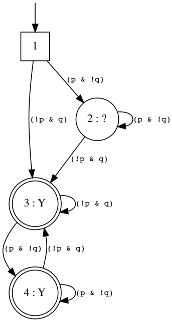
[7] K. Schneider. *Improving Automata Generation for Linear Temporal Logic by Considering the Automaton Hierarchy.* (LPAR 2001)

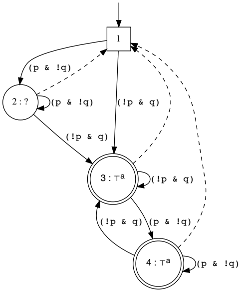
Structure of Explicit-State Monitors (1) - $p \,\mathsf{U}\, q$

Monitor Levels for Optimization Purposes:

- **L1** The monitor synthesis stops at all conclusive states;
- **L2** The monitor synthesis explores all states;
- **L3** The monitor synthesis explores all states and reset states.

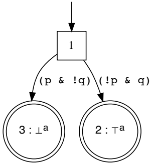


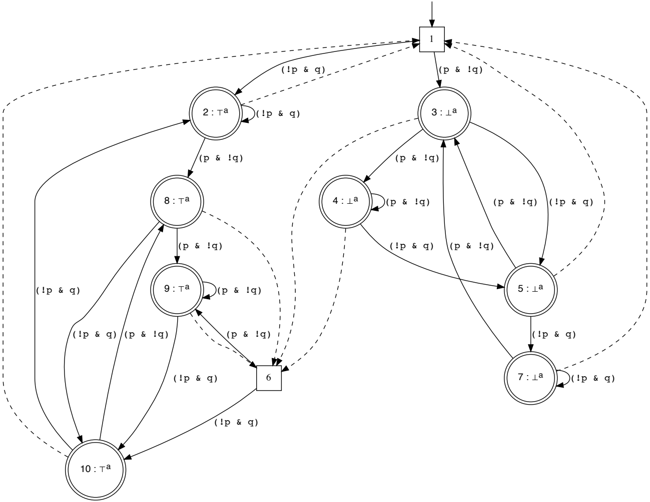




Structure of Explicit-State Monitors (2)

The "Iceberg" of Y $p \vee q$:





ABRV-LTL verdicts (and the lattice)

$\mathbb{B}_4 \doteq \{\top^{\mathrm{a}}, \bot^{\mathrm{a}}, ?, \times\}$:

- *conclusive true* ($\top^{\mathrm{a}}$) (or *true under assumption*)
- *conclusive false* ($\bot^{\mathrm{a}}$) (or *false under assumption*)
- *inconclusive* ($?$)
- *out-of-model* ($\times$)

The lattice

$$\begin{tikzcd} \top \\ ? \arrow[u] \arrow[d] \\ \bot \end{tikzcd}$$

LTL$_3$

$$\begin{tikzcd} & \times & \\ \top^{\mathrm{a}} \arrow[ur] & & \bot^{\mathrm{a}} \arrow[ul] \\ & ? \arrow[ul] \arrow[ur] & \end{tikzcd}$$

ABRV-LTL

ABRV-LTL semantics

Let $K \doteq \langle V_K, \Theta_K, \rho_K, \mathcal{J}_K \rangle$ be an fks, φ be an LTL formula built from AP. Let $\psi(O) \in \Psi(O)^*$ be a finite sequence of Boolean formulae over $O \subseteq V_K \cup AP$. We also define

$$\mathcal{L}^K(\psi(O)) \doteq \left\{ w \in \mathcal{L}(K) \;\middle|\; \forall i.\, i < |\psi(O)| \Rightarrow w_i(V_K \cup AP) \models \psi_i(O) \right\}$$

to be the set of runs in K which are compatible with $\psi(O)$.

Definition

The ABRV-LTL semantics of φ over $\psi(O)$ under K is defined as

$$[\![\psi(O), i \models \varphi]\!]_4^K \doteq \begin{cases} \times, & \text{if } \mathcal{L}^K(\psi(O)) = \emptyset \\ \top^{\mathrm{a}}, & \text{if } \mathcal{L}^K(\psi(O)) \neq \emptyset \wedge \forall w \in \mathcal{L}^K(\psi(O)).\; w, i \models \varphi \\ \bot^{\mathrm{a}}, & \text{if } \mathcal{L}^K(\psi(O)) \neq \emptyset \wedge \forall w \in \mathcal{L}^K(\psi(O)).\; w, i \models \neg\varphi \\ ?, & \text{otherwise} \;. \end{cases}$$

Basic Optimizations (2)

if o_3 **then** $F := \texttt{ltl\_translation}((\bigwedge_{\psi \in \mathcal{J}_K \cup \mathcal{J}_\varphi} \mathrm{GF}\,\psi) \to \text{false})$;

function `check_nonemptiness(r)`

  if $o_2 \wedge (\texttt{SMT}(r) = \text{unsat})$ **then return** $\perp$;

  else

    return $\neg\texttt{model\_checking}(\langle V, r, \rho_K \wedge \rho_\varphi, \mathcal{J}_K \cup \mathcal{J}_\varphi \rangle,\ o_3\ ?\ F\ :\ \text{false})$

function `model_checking(M, ψ)`

  if o_4 **then**

    if $\texttt{BMC}(M, \psi) = \perp$ **then return** $\perp$; // counter-example found

    else // max_k reached

      return $\texttt{IC3\_IA}(M, \psi)$

  else return $\texttt{IC3\_IA}(M, \psi)$;

Incremental BMC: The algorithm (1 of 3)

New procedures (API)

- `init_nonemptiness` for creating a persistent SMT solver instance,
- `update_nonemptiness` for checking nonemptiness of belief states after new observation,
- `reset_nonemptiness` for resetting the SMT solver, cleaning up all existing observations.

function `init_nonemptiness`(I, T)
    e := new BMC solver with initial formula I and transition relation T
    `reset_nonemptiness`(e, I)
    return e

procedure `reset_nonemptiness`(e, I)
    $e.problem := I(V_0)$; // the initial formula unrolled at time 0
    $e.observations := []$; // an array holding observations
    $e.n := 0$; // the number of observations
    $e.map := \{\}$; // a hash map from time to (unused) observations
    $e.k := 0$; // the number of unrolled transition relations
    $e.min\_k := 0$; // internal parameter, to be updated by $e.observations$
    $e.max\_k := max\_k$; // a local copy of max_k

Incremental BMC: The algorithm (2 of 3)

function $\texttt{bmc\_monitor}(K \doteq \langle V_K, \Theta_K, \rho_K, \mathcal{J}_K \rangle,\ \varphi,\ u,\ max\_k,\ window\_size)$

$T_\varphi \doteq \langle V_\varphi, \Theta_\varphi, \rho_\varphi, \mathcal{J}_\varphi \rangle := \texttt{ltl\_translation}(\varphi)$

$T_{\neg\varphi} \doteq \langle V_\varphi, \Theta_{\neg\varphi}, \rho_\varphi, \mathcal{J}_\varphi \rangle := \texttt{ltl\_translation}(\neg\varphi)$

$V := V_K \cup V_\varphi$

$e_1 := \texttt{init\_nonemptiness}(\Theta_K \wedge \Theta_\varphi, \rho_K \wedge \rho_\varphi)$

$e_2 := \texttt{init\_nonemptiness}(\Theta_K \wedge \Theta_{\neg\varphi}, \rho_K \wedge \rho_\varphi)$

for $0 < i < |u|$ **do**

$b_1 := \texttt{update\_nonemptiness}(e_1, u_i)$

$b_2 := \texttt{update\_nonemptiness}(e_2, u_i)$

if $b_1 \wedge b_2$ **then return** ? ; `// inconclusive`

else if b_1 **then return** $\top^a$; `// conditionally true`

else if b_2 **then return** $\bot^a$; `// conditionally false`

else return $\times$; `// out of model`

function $\texttt{compute\_belief\_states}(e)$

$r := e.I(V)$

for $i \leftarrow 0$ **to** $e.n$ **do**

if $i = 0$ **then** $r := r \wedge e.observations[i](V)$;

else

$r := \texttt{quantifier\_elimination}(V, r \wedge T(V, V')) \wedge e.observations[i](V)$

return r

Incremental BMC: The algorithm (3 of 3)

```
function update_nonemptiness(e, o)
    e.map[e.n] = o,    e.observations[e.n++] = o;                 // store new observation
    for (k, v) : e.map do
        if k ⩽ e.k then  e.problem := e.problem ∧ v(V_i)
        delete e.map[k] ;
        if k > e.min_k then
            e.min_k := k ;                    // set to the maximal time of observations
    result := ?
    while e.k ⩽ e.max_k and result = ? do
        i := e.k
        if SMT(e.problem) = unsat then  result := ⊥, break;
        if e.k ⩾ e.min_k and SMT(e.problem ∧ [[F]]_i) = sat then  result = ⊤, break  ;
        e.problem := e.problem ∧ e.T(V_i, V_{i+1})
        if e.map[i + 1] exists  then
            e.problem := e.problem ∧ e.map[i + 1](V_{i+1}),    delete e.map[i + 1]
        e.k := e.k + 1
    e.max_k := e.max_k + 1;                   // increase the search bound for next calls
    if e.k > window_size or result = ? then
        r := compute_belief_states(e); reset_nonemptiness(e, r)
    if result = ⊤ or result = ⊥ then
        return result
    else
        return ¬IC3_IA(⟨V, r, e.T, 𝒥_K ∪ 𝒥_φ⟩, false)
```

Unboundedness of Infinite-State Monitors

Cantor's Ternary Set, initially we have one interval $[a, b]$ where $a = 0,\ b = 1$

$$[a, a + (b - a)/3], \quad [a + (b - a)/3, a + 2(b - a)/3], \quad [a + 2(b - a)/3, b]$$

- At time 0, there is only one interval $[0, 1]$;
- At time 1, there are two intervals: $[0, \frac{1}{3}]$ and $[\frac{2}{3}, 1]$;
- At time 2, there are four intervals: $[0, \frac{1}{9}]$, $[\frac{2}{9}, \frac{1}{3}]$, $[\frac{2}{3}, \frac{7}{9}]$, $[\frac{8}{9}, 1]$.
- At time 3, there are 8 intervals: $[0, \frac{1}{27}]$, $[\frac{2}{27}, \frac{1}{9}]$, $[\frac{2}{9}, \frac{7}{27}]$, $[\frac{8}{27}, \frac{1}{3}]$, $[\frac{2}{3}, \frac{18}{27}]$, $[\frac{20}{27}, \frac{7}{9}]$, $[\frac{8}{9}, \frac{25}{27}]$, $[\frac{26}{27}, 1]$;
- At time 4, there are 16 intervals: ...

A transition system choosing next interval non-deterministically

$V \doteq \{a, b, x\}$; $\quad \Theta \doteq (a = 0 \wedge b = 1)$; $\quad \rho \doteq (a' = a \wedge b' = a + (b - a)/3) \vee (a' = a + 2(b - a)/3 \wedge b' = b)$.

The monitoring property is $\mathrm{G}\,\neg(a \leqslant x \wedge x \leqslant b)$ (x is NOT in any of the intervals).

MC reduced to ABRV (impossible without assumptions)

Theorem

Let K be a model (as an FTS), and φ be a property in temporal logics like LTL. The model checking problem $K \models \varphi$ can be done by ABRV monitoring on empty traces using the same model as RV assumptions.

Proof.

By definitions of ABRV monitor and ABRV-LTL we have

$$\mathcal{M}_\varphi^K(\epsilon) = \begin{cases} \top^{\mathrm{a}}, & \text{if } [\![K \models \varphi]\!] = \top \text{ (and } [\![K \models \neg\varphi]\!] = \bot\text{)}, \\ \bot^{\mathrm{a}}, & \text{if } [\![K \models \varphi]\!] = \bot \text{ (and } [\![K \models \neg\varphi]\!] = \top\text{)}, \\ ?\,, & \text{if } [\![K \models \varphi]\!] = [\![K \models \neg\varphi]\!] = \bot \text{ (counterexamples exist on both sides)}, \\ \times, & \text{if } [\![K \models \varphi]\!] = [\![K \models \neg\varphi]\!] = \top \text{ (i.e. } \mathcal{L}(K) = \emptyset\text{, i.e. } K \text{ is an empty model)}. \end{cases}$$

Thus $[\![K \models \varphi]\!] = \top$ iff $\mathcal{M}_\varphi^K(\epsilon) = \top^{\mathrm{a}}$ (or $\times$ if the model K is empty). □

ABRV reduced to MC

Theorem

Let K be RV assumptions (as FTS), and φ be a monitoring property in temporal logics like LTL. Let $u = s_0 \ldots s_{n-1}$ be a finite trace (thus $|u| = n$). The ABRV monitoring problem $\mathcal{M}_\varphi^K(u)$ can be done by two calls of model checking on combined models from K and u.

Proof.

Let c be a fresh integer variable taking finite domain values from 0 to $n-1$. Let $S_u = \langle V_k \cup \{c\}, \Theta, \rho, \emptyset \rangle$ be a Kripke Structure built from u, where

$$\Theta \doteq (c = 0) \wedge s_0,$$

$$\rho \doteq (c' = \min\{c+1, n\}) \wedge \bigwedge_{i=1}^{n-1} ((c = i) \rightarrow s_i)$$

Then, $\mathcal{M}_\varphi^K(u)$ can be computed by two MC calls $[\![K \times S_u \models \varphi]\!]$ and $[\![K \times S_u \models \neg\varphi]\!]$. $\square$

$[\![K \times S_u \models \varphi]\!]$	$[\![K \times S_u \models \neg\varphi]\!]$	$\mathcal{M}_\varphi^K(u)$
$\top$	$\top$	$\times$
$\top$	$\bot$	$\top$[a]
$\bot$	$\top$	$\bot$[a]
$\bot$	$\bot$	?

ABRV of Infinite-State Systems is Undecidable

Proof.

1. Combining results from the previous two slides, ABRV and MC have the same (worst-case) space and time complexities (there exist bidirectional reductions);
2. Such a close relationship betwen ABRV and MC does not hold for traditional RV without assumptions;
3. MC of infinite-state systems is in general *undecidable*, thus so is ABRV (of infinite-state systems).

□

(NOTE: furthermore, even with decidable inputs, ABRV of infinite-state systems may not consume bounded resources w.r.t. input size. To be explained.)

ABRV reduced to MC and QE (2)

```
function monitor2(K ≐ ⟨V_K, Θ_K, ρ_K, 𝒥_K⟩, φ, u)
    T_φ  ≐ ⟨V_φ, Θ_φ , ρ_φ, 𝒥_φ⟩ := ltl_translation(φ)
    T_¬φ ≐ ⟨V_φ, Θ_¬φ, ρ_φ, 𝒥_φ⟩ := ltl_translation(¬φ)
    V := V_K ∪ V_φ
    ⟨r_φ, r_¬φ⟩ := ⟨Θ_K ∧ Θ_φ, Θ_K ∧ Θ_¬φ⟩
    if |u| > 0 then
        ⟨r_φ, r_¬φ⟩ := ⟨r_φ ∧ u_0, r_¬φ ∧ u_0⟩
    for 1 ⩽ i < |u| do
        r_φ  := quantifier_elimination(V, ρ_K ∧ ρ_φ ∧ r_φ ) ∧ u_i
        r_¬φ := quantifier_elimination(V, ρ_K ∧ ρ_φ ∧ r_¬φ) ∧ u_i
    b_1 := ¬model_checking(⟨V, r_φ, ρ_K ∧ ρ_φ, 𝒥_K ∪ 𝒥_φ⟩, false)
    b_2 := ¬model_checking(⟨V, r_¬φ, ρ_K ∧ ρ_φ, 𝒥_K ∪ 𝒥_φ⟩, false)
    if b_1 ∧ b_2 then return ? ;                    // inconclusive
    else if b_1 then return ⊤^a;                    // conditionally true
    else if b_2 then return ⊥^a;                    // conditionally false
    else return ×;                                  // out of model
```