Assumption-Based Runtime Verification of Infinite-State Systems

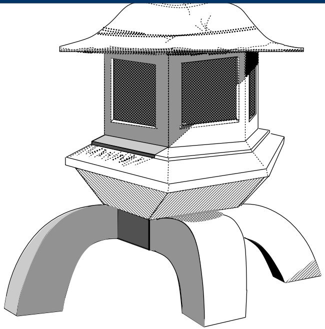
(RV 2021 regular paper)

Alessandro Cimatti [1]
Chun Tian [12]
Stefano Tonetta [1]
{cimatti,ctian,tonettas}@fbk.eu

[1]Fondazione Bruno Kessler, Italy

[2]University of Trento, Italy

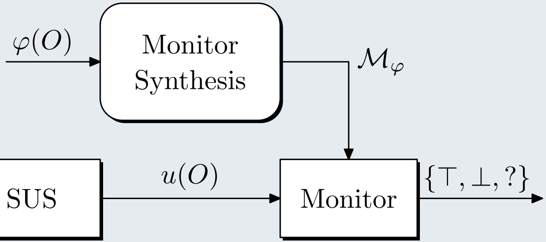
October 12, 2021



Outline

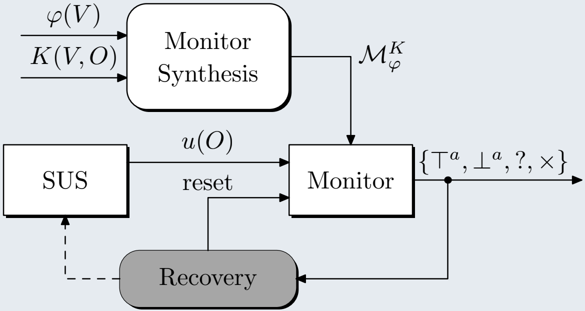
Runtime Verification (RV)

- Runtime monitors synthesized from a specification (or property);
- The monitor returns verdicts indicating if the input trace violates the property or not.



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.



ABRV of Infinite-State Systems

In 2019, the authors have given BDD-based algorithms[a] and tools[b] for ABRV of *finite-state systems*. The present work is about ABRV of *infinite-state systems*.

---

[a] 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

[b] 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

Main ideas

- Runtime Verification (RV) reduced to Model Checking (MC);
- First-Order Quantifier Elimination (QE) for forward image computation;
- Fast (incomplete) Bounded Model Checking (BMC) before full MC algorithms;
- Incremental BMC.

Outline

Preliminaries (2)

Fair Transition System (FTS)

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

where $V = \{x_1, \ldots, x_n\}$ is a finite set of variables, Θ the *initial condition*, ρ the *transition relation*, and $\mathcal{J}$ a (finite) set of *justice conditions*. (Θ, ρ and each element of $\mathcal{J}$ are quantifier-free $\mathcal{T}$-formulas.)

Linear Temporal Logic (LTL) or LTL Modulo Theory

$$\varphi ::= \text{true} \mid \alpha \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \mathbf{Y}\varphi \mid \varphi\,\mathbf{S}\,\varphi$$

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 $\mathcal{T}$.

Preliminaries (3, ABRV)

Set of fair paths

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

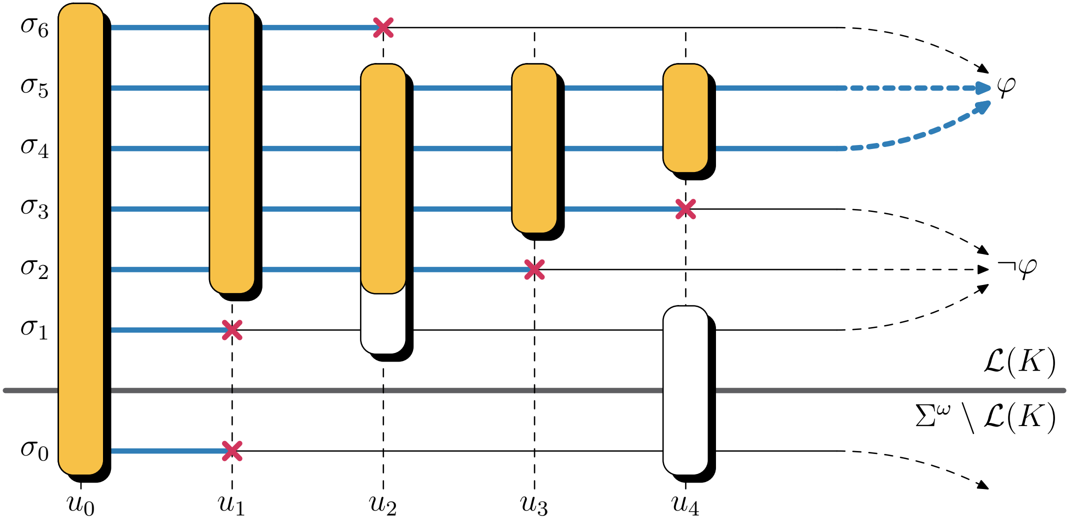
ABRV-LTL semantics

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

ABRV monitor

$$\mathcal{M}_\varphi^K(u) \doteq [\![u, 0 \models \varphi]\!]_4^K \quad .$$

Preliminaries (4, ABRV illustration)



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

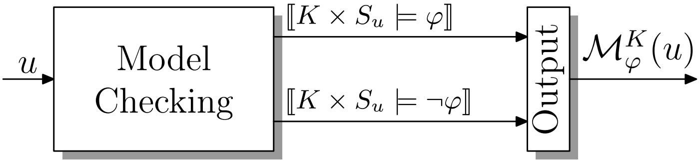
Outline

ABRV reduced to Model Checking

Let c be a integer variable, $S_u = \langle V_k \cup \{c\}, \Theta, \rho, \emptyset \rangle$, where

$$\Theta = (c = 0) \wedge u_0, \qquad \rho = \bigwedge_{i=0}^{|u|-1} \left((c = i) \rightarrow (c' = i + 1 \wedge u'_{i+1})\right)$$

$\mathcal{M}_\varphi^K(u)$ can be computed by two MC calls:



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

Model Checking reduced to ABRV

Computing $[\![K \models \varphi]\!]$ by ABRV monitoring

Let ϵ be an empty trace, by ABRV definition we have

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

Thus $[\![K \models \varphi]\!] = \top$ iff $\mathcal{M}_\varphi^K(\epsilon) = \top^{\mathrm{a}}$ or $\times$ (usually the model in a MC problem is not empty).

Conclusion

ABRV with infinite-state assumptions is undecidable (because infinite-state MC is known to be undecidable.)

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

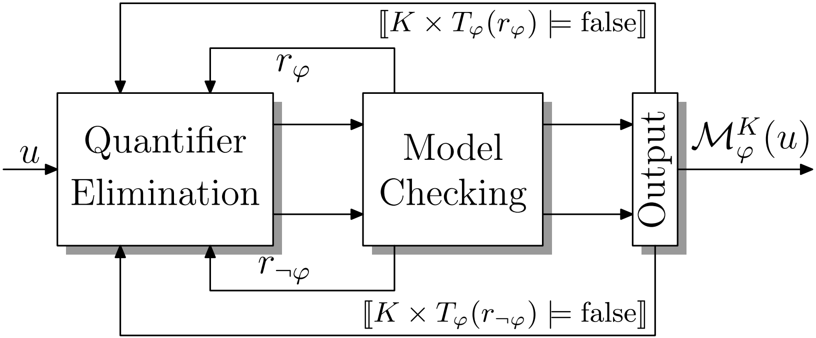
$$\mathrm{fwd}(\psi(V), \rho(V, V'))(V) \doteq \big(\exists V.\ \rho(V, V') \wedge \psi(V)\big)[V/V']$$

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]
$\bot$	$\bot$	$\times$

Outline

Basic Optimizations (1)

Basic optimization ideas

- o1 If the monitor has already reached conclusive verdicts ($\top^{\mathrm{a}}$ or $\bot^{\mathrm{a}}$), 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 plain BMC (or any other MC procedure which only detects counterexamples) before calling a unbounded model checker such as IC3IA.

Theorem

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

Incremental BMC: The Idea

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

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

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 algorithm (1)

New procedures

- `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`
  $e.max\_k := max\_k$; `// a local copy of` max_k

Incremental BMC: The algorithm (2)

```
function bmc_monitor(K ≐ ⟨V_K, Θ_K, ρ_K, 𝒥_K⟩, φ, u, max_k, window_size)
    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
    e_1 := init_nonemptiness(Θ_K ∧ Θ_φ , ρ_K ∧ ρ_φ)
    e_2 := init_nonemptiness(Θ_K ∧ Θ_¬φ, ρ_K ∧ ρ_φ)
    for 0 < i < |u| do
        b_1 := update_nonemptiness(e_1, u_i)
        b_2 := update_nonemptiness(e_2, u_i)
    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
```

Incremental BMC: The algorithm (3)

function `update_nonemptiness`(e, o)

$e.map[e.n] = o, \quad e.observations[e.n++] = o;$ `// store new observation`

for $(k, v) : e.map$ **do**

  if $k \leqslant e.k$ **then** $e.problem := e.problem \wedge v(V_i)$

  delete $e.map[k]$;

$result := ?$

while $e.k \leqslant e.max\_k$ **and** $result = ?$ **do**

  $i := e.k$

  if $\mathtt{SMT}(e.problem) = \textsc{unsat}$ **then** $result := \bot$, **break**;

  if $\mathtt{SMT}(e.problem \wedge [[F]]_i) = \textsc{sat}$ **then** $result = \top$, **break**;

  $e.problem := e.problem \wedge e.T(V_i, V_{i+1})$

  if $e.map[i+1]$ *exists* **then**

    $e.problem := e.problem \wedge e.map[i+1](V_{i+1}), \quad$ delete $e.map[i+1]$

  $e.k++$

$e.max\_k++;$ `// 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 $result = \top$ **or** $result = \bot$ **then**

  return *result*

else

  return $\neg$`IC3_IA`$(\langle V, r, e.T, \mathcal{J}_K \cup \mathcal{J}_\varphi\rangle, \mathrm{false})$

Incremental BMC: The algorithm (4)

function `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
      $r :=$ `quantifier_elimination`$(V, r \wedge T(V, V')) \wedge e.observations[i](V)$
  return r

Tool Implementation

The RV algorithms presented in this paper have been implemented in NuRV since version 1.6.0 (`https://es.fbk.eu/tools/nurv/`). (Currently we support 5 operating systems, 5 target languages for monitor code generation, and network-based monitoring.)

Outline

Performance Tests on Dwyer's LTL patterns (1)

Pattern 49

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

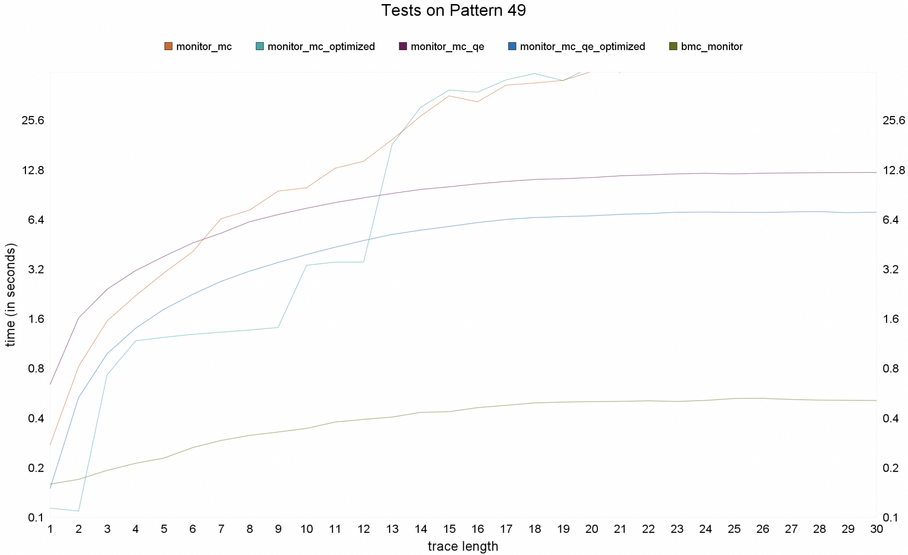
RV assumptions

"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)



Outline

Conclusions

Conclusions

- The ABRV framework has been extended in this paper to assumptions defined as infinite-state system, using existing SMT, MC, QE techniques.
- We start from a trivial reduction from RV to MC, and eventually obtained an highly optimized RV algorithm, based on Incremental BMC. (The final version is hundreds of times faster than the initial one.)

Possible concerns of the present approach

- The use of (slow) SMT solvers in performing (fast) runtime monitoring; (there is a trade-off between the required speed of the monitor and the complexity of the assumptions)
- The boundedness of memory consumptions during runtime monitoring. (unbound in the worst-case but better in practical)

Some Future Directions

- Better performances (including enginering level changes, e.g. 2x faster after paper submission);
- More thoroughly tests on better RV benchmarks;
- More compact presentations of raw formulas (DBM, PPlite, ...);
- Attacking real-time temporal properties with timed assumptions;
- Code generation for infinite-state monitors (e.g. calling external SMT solvers by standard interface like DIMACS).

Preliminaries (1)

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 $\mathcal{T}$.
SMT is the basis of infinite-state model checking algorithms used in this paper. Note: *Tool implementation and experimental evaluations are based 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.
MathSAT supports two QE procedures: Fourier-Motzkin and Loos-and-Weispfenning ($\mathcal{LRA}$).

ABRV reduced to Model Checking (2)

function `monitor1`$(K \doteq \langle V_K, \Theta_K, \rho_K, \mathcal{J}_K \rangle, \varphi, u)$

- $\Theta := \top$, $\rho := \top$
- **if** $|u| > 0$ **then**
  - $\Theta := (c = 0) \wedge u_0$
  - **if** $|u| > 1$ **then**
    - $\rho := \bigwedge_{i=0}^{|u|-1} ((c = i) \rightarrow (c' = i + 1 \wedge u'_{i+1}))$
- $S_u := \langle V_k \cup \{c\}, \Theta, \rho, \emptyset \rangle$
- $b_1 :=$ `model_checking`$(K \times S_u,\ \varphi)$
- $b_2 :=$ `model_checking`$(K \times S_u,\ \neg\varphi)$
- **if** $b_1 \wedge b_2$ **then return** $\times$; `// out of model`
- **else if** b_1 **then return** $\top^{\mathrm{a}}$; `// conditionally true`
- **else if** b_2 **then return** $\bot^{\mathrm{a}}$; `// conditionally false`
- **else return** ?; `// inconclusive`

ABRV reduced to MC and QE (2)

$$
\begin{array}{l}
\textbf{function}\ \texttt{monitor2}(K \doteq \langle V_K, \Theta_K, \rho_K, \mathcal{J}_K \rangle, \varphi, u) \\
\quad T_\varphi \doteq \langle V_\varphi, \Theta_\varphi, \rho_\varphi, \mathcal{J}_\varphi \rangle := \texttt{ltl\_translation}(\varphi) \\
\quad T_{\neg\varphi} \doteq \langle V_\varphi, \Theta_{\neg\varphi}, \rho_\varphi, \mathcal{J}_\varphi \rangle := \texttt{ltl\_translation}(\neg\varphi) \\
\quad V := V_K \cup V_\varphi \\
\quad \langle r_\varphi, r_{\neg\varphi} \rangle := \langle \Theta_K \wedge \Theta_\varphi, \Theta_K \wedge \Theta_{\neg\varphi} \rangle \\
\quad \textbf{if}\ |u| > 0\ \textbf{then} \\
\quad\quad \langle r_\varphi, r_{\neg\varphi} \rangle := \langle r_\varphi \wedge u_0, r_{\neg\varphi} \wedge u_0 \rangle \\
\quad \textbf{for}\ 1 \leqslant i < |u|\ \textbf{do} \\
\quad\quad r_\varphi := \texttt{quantifier\_elimination}(V, \rho_K \wedge \rho_\varphi \wedge r_\varphi) \wedge u_i \\
\quad\quad r_{\neg\varphi} := \texttt{quantifier\_elimination}(V, \rho_K \wedge \rho_\varphi \wedge r_{\neg\varphi}) \wedge u_i \\
\quad b_1 := \neg\texttt{model\_checking}(\langle V, r_\varphi, \rho_K \wedge \rho_\varphi, \mathcal{J}_K \cup \mathcal{J}_\varphi \rangle, \mathrm{false}) \\
\quad b_2 := \neg\texttt{model\_checking}(\langle V, r_{\neg\varphi}, \rho_K \wedge \rho_\varphi, \mathcal{J}_K \cup \mathcal{J}_\varphi \rangle, \mathrm{false}) \\
\quad \textbf{if}\ b_1 \wedge b_2\ \textbf{then return}\ ?\,; \hfill \texttt{// inconclusive} \\
\quad \textbf{else if}\ b_1\ \textbf{then return}\ \top^{\mathrm{a}}; \hfill \texttt{// conditionally true} \\
\quad \textbf{else if}\ b_2\ \textbf{then return}\ \bot^{\mathrm{a}}; \hfill \texttt{// conditionally false} \\
\quad \textbf{else return}\ \times; \hfill \texttt{// out of model}
\end{array}
$$

Basic Optimizations (2)

```
if o3 then  F := ltl_translation((⋀_{ψ∈J_K∪J_φ} GF ψ) → false) ;
function check_nonemptiness(r)
    if o2 ∧ (SMT(r) = unsat) then return ⊥ ;
    else
        return ¬model_checking(⟨V, r, ρ_K ∧ ρ_φ, J_K ∪ J_φ⟩, o3 ? F : false)
function model_checking(M, ψ)
    if o4 then
        if BMC(M, ψ) = ⊥ then return ⊥;                // counterexample found
        else                                              // max_k reached
            return IC3_IA(M, ψ)
    else return IC3_IA(M, ψ);
```