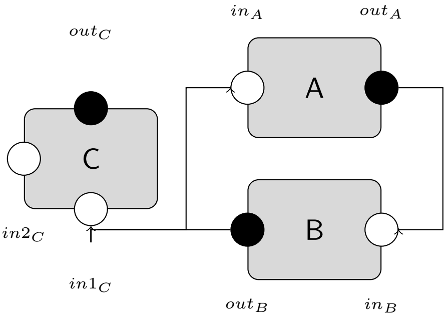
Compositional reasoning over asynchronous systems: a temporal logic-based approach

Alberto Bombardelli Stefano Tonetta

Fondazione Bruno Kessler

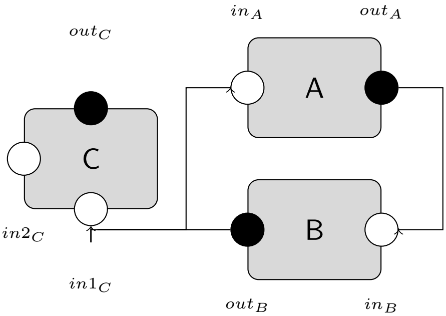
Introduction

Interface Component



- Each component has input and output ports
- Ports of different components can be connected together
- Composition is potentially hierarchical

Interface Component



- Each component has input and output ports
- Ports of different components can be connected together
- Composition is potentially hierarchical

Leaf components: $\mathcal{M} = \langle \mathcal{V}^I, \mathcal{V}^O, \mathcal{I}, \mathcal{T}, \mathcal{F} \rangle$
Represented by Interface Transition Systems

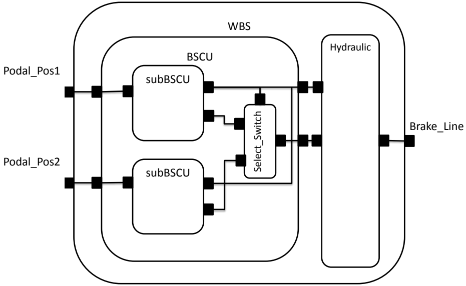
Composite components:
$\mathcal{M} = \mathcal{M}_1 \otimes \cdots \otimes \mathcal{M}_n$
Represented by the composition of components (both composite and leaf) where $\otimes$ is the composition operator

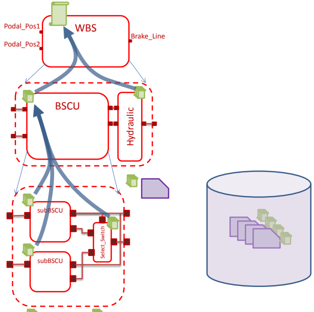
Contract based design with OCRA

- Consider for each component C a contract $\langle A, G \rangle$: couple of assumption and guarantee LTL properties
- Defines a notion of refinement: a component C is refined by its sub-components $Sub(C) = \{C_1, \ldots C_n\}$ iff
  1. Impl: $(\bigwedge_{C_i \in Sub(C)} (A_i \rightarrow G_i)) \rightarrow (A \rightarrow G)$
  2. Env: For all $C_i \in Sub(C)$: $(\bigwedge_{C_j \in Sub(C) \setminus \{C_i\}} (A_j \rightarrow G_j)) \rightarrow (A \rightarrow A_i)$
  3. For each leaf component of the hierarchy: $C_{leaf} \models A_{leaf} \rightarrow G_{leaf}$

Contract based design with OCRA

- Consider for each component C a contract $\langle A, G \rangle$: couple of assumption and guarantee LTL properties
- Defines a notion of refinement: a component C is refined by its sub-components $Sub(C) = \{C_1, \dots C_n\}$ iff
  1. Impl: $(\bigwedge_{C_i \in Sub(C)} (A_i \rightarrow G_i)) \rightarrow (A \rightarrow G)$
  2. Env: For all $C_i \in Sub(C)$: $(\bigwedge_{C_j \in Sub(C) \setminus \{C_i\}} (A_j \rightarrow G_j)) \rightarrow (A \rightarrow A_i)$
  3. For each leaf component of the hierarchy: $C_{leaf} \models A_{leaf} \rightarrow G_{leaf}$



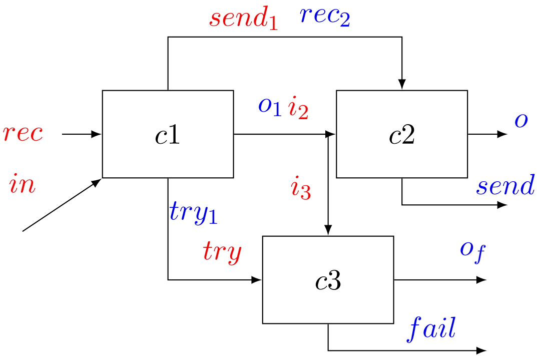


Asynchronous composition

Introducing asynchrony

- What if components runs asynchronously? Interleaving execution of components.
- We introduce a special boolean variable run_i that is true iff C_i is running.
- We introduce the scheduling constraint α.
- We expect each component C_i to run infinitely often (encoded inside α).
- Output variables ($\mathcal{V}_i^O$) do not change when C_i is not running

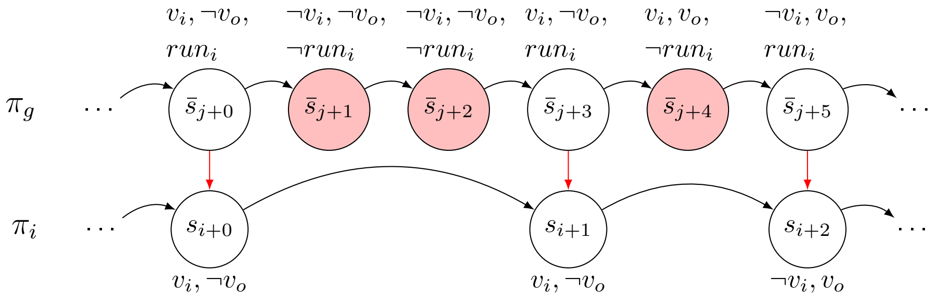
Toy example



$$\varphi_{c1} := \mathbf{G}(rec \rightarrow o_1' = i \wedge$$
$$(try_1 \wedge o_1' = o_1)\mathbf{U} send_1)$$
$$\varphi_{c2} := \mathbf{G}(rec_2 \rightarrow o' = i_2 \wedge \mathbf{X} send_2)$$
$$\varphi_{c3} := \mathbf{G}(try \rightarrow o_f' = i_3 \wedge \mathbf{X} fail)$$
$$\varphi := \mathbf{G}(rec \wedge i = v \rightarrow$$
$$\mathbf{F}(send \wedge o = v \vee fail \wedge o_f = v))$$
$$\alpha := \mathbf{G}(i_1 \rightarrow run_1) \wedge$$
$$\mathbf{G}(send_1 \rightarrow run_2) \wedge$$
$$\mathbf{G}(\mathbf{H}_{\leq p} try_1 \rightarrow run_3)$$

Trace projection

For each trace π_g of the composition, we consider the projected trace to the local component π_i:



LTL rewriting

We define a rewriting $\mathcal{R}^*$ such that

$$\pi_{local} \models \varphi_i \Leftrightarrow \pi_{global} \models \mathcal{R}^*(\varphi_i)$$

LTL rewriting

We define a rewriting $\mathcal{R}^*$ such that

$$\pi_{local} \models \varphi_i \Leftrightarrow \pi_{global} \models \mathcal{R}^*(\varphi_i)$$

$\mathcal{R}$ it assumes to be on a state in which run_i holds

- a: $\mathcal{R}(a) := a$
- $\neg$: $\mathcal{R}(\neg\psi) := \neg\mathcal{R}(\psi)$
- $\vee$: $\mathcal{R}(\psi_1 \vee \psi_2) := \mathcal{R}(\psi_1) \vee \mathcal{R}(\psi_2)$
- **X**: $\mathcal{R}(\mathbf{X}\psi) := \mathbf{X}(run_i\mathbf{R}(\neg run_i \vee \mathcal{R}(\psi)))$
- **U**: $\mathcal{R}(\psi_1\mathbf{U}\psi_2) := (\neg run_i \vee \mathcal{R}(\psi_1))\mathbf{U}(run_i \wedge \mathcal{R}(\psi_2))$

LTL rewriting

We define a rewriting $\mathcal{R}^*$ such that

$$\pi_{local} \models \varphi_i \Leftrightarrow \pi_{global} \models \mathcal{R}^*(\varphi_i)$$

$\mathcal{R}$ it assumes to be on a state in which run_i holds

- a: $\mathcal{R}(a) := a$
- $\neg$: $\mathcal{R}(\neg\psi) := \neg\mathcal{R}(\psi)$
- $\vee$: $\mathcal{R}(\psi_1 \vee \psi_2) := \mathcal{R}(\psi_1) \vee \mathcal{R}(\psi_2)$
- **X**: $\mathcal{R}(\mathbf{X}\psi) := \mathbf{X}(run_i\mathbf{R}(\neg run_i \vee \mathcal{R}(\psi)))$
- **U**: $\mathcal{R}(\psi_1\mathbf{U}\psi_2) := (\neg run_i \vee \mathcal{R}(\psi_1))\mathbf{U}(run_i \wedge \mathcal{R}(\psi_2))$
- **Y**: $\mathcal{R}(\mathbf{Y}\psi) := \mathbf{Y}(\neg run_i\mathbf{S}(run_i \wedge \mathcal{R}(\psi)))$
- **S**: $\mathcal{R}(\psi_1\mathbf{S}\psi_2) := (\neg run_i \vee \mathcal{R}(\psi_1))\mathbf{S}(run_i \wedge \mathcal{R}(\psi_2))$
- ...

LTL rewriting

We define a rewriting $\mathcal{R}^*$ such that

$$\pi_{local} \models \varphi_i \Leftrightarrow \pi_{global} \models \mathcal{R}^*(\varphi_i)$$

$\mathcal{R}$ it assumes to be on a state in which run_i holds

- a: $\mathcal{R}(a) := a$
- $\neg$: $\mathcal{R}(\neg\psi) := \neg\mathcal{R}(\psi)$
- $\vee$: $\mathcal{R}(\psi_1 \vee \psi_2) := \mathcal{R}(\psi_1) \vee \mathcal{R}(\psi_2)$
- **X**: $\mathcal{R}(\mathbf{X}\psi) := \mathbf{X}(run_i\mathbf{R}(\neg run_i \vee \mathcal{R}(\psi)))$
- **U**: $\mathcal{R}(\psi_1\mathbf{U}\psi_2) := (\neg run_i \vee \mathcal{R}(\psi_1))\mathbf{U}(run_i \wedge \mathcal{R}(\psi_2))$
- **Y**: $\mathcal{R}(\mathbf{Y}\psi) := \mathbf{Y}(\neg run_i\mathbf{S}(run_i \wedge \mathcal{R}(\psi)))$
- **S**: $\mathcal{R}(\psi_1\mathbf{S}\psi_2) := (\neg run_i \vee \mathcal{R}(\psi_1))\mathbf{S}(run_i \wedge \mathcal{R}(\psi_2))$
- ...

$\mathcal{R}^*$ "maps" 0 to the first transition with run_i

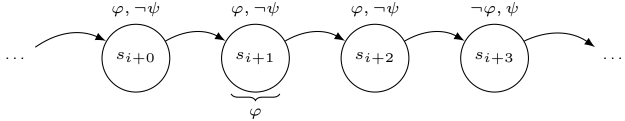
$$\mathcal{R}^*(\varphi_i) = run_i\mathbf{R}(\neg run_i \vee \mathcal{R}(\varphi_i))$$

Example neXt

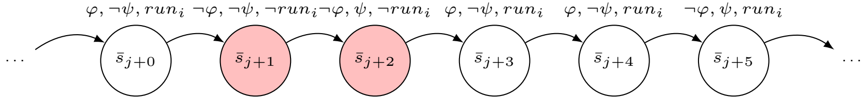
$Prop_{loc} = \mathbf{X}\varphi$

$Prop_{glob} = \mathcal{R}(\mathbf{X}\varphi) = \mathbf{X}(run_i \mathbf{R}(st \vee \varphi))$

π_{local}



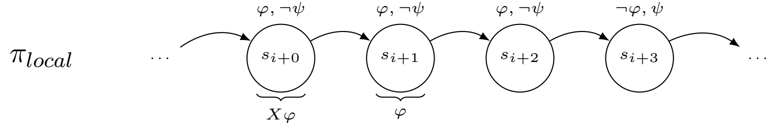
π_{global}

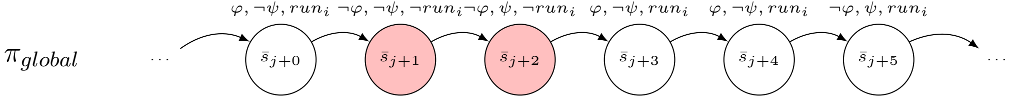


Example neXt

$Prop_{loc} = \mathbf{X}\varphi$

$Prop_{glob} = \mathcal{R}(\mathbf{X}\varphi) = \mathbf{X}(run_i \mathbf{R}(st \vee \varphi))$

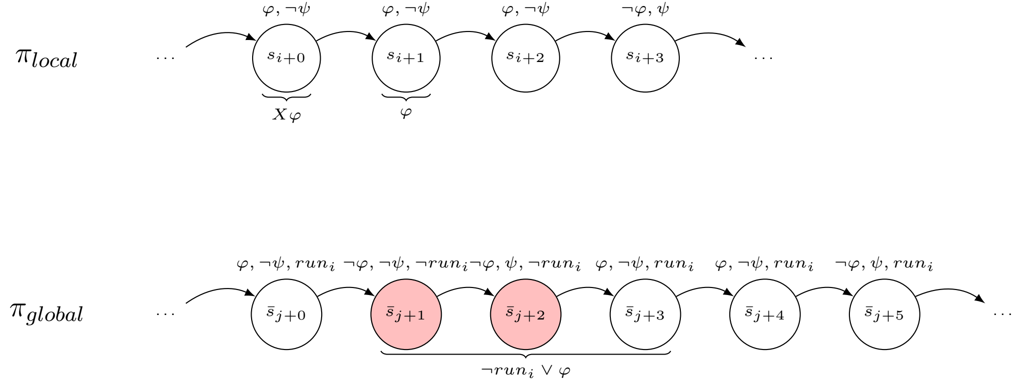




Example neXt

$Prop_{loc} = \mathbf{X}\varphi$

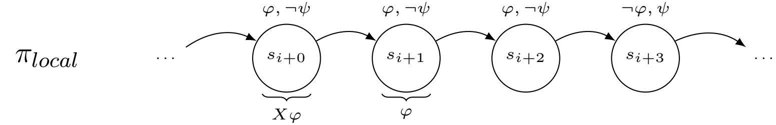
$Prop_{glob} = \mathcal{R}(\mathbf{X}\varphi) = \mathbf{X}(run_i \mathbf{R}(st \vee \varphi))$

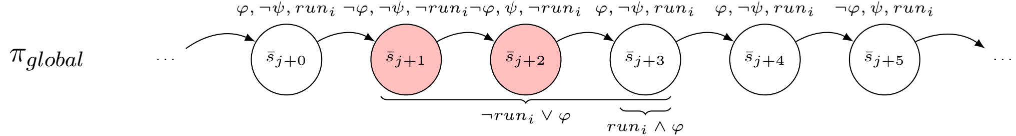


Example neXt

$Prop_{loc} = \mathbf{X}\varphi$

$Prop_{glob} = \mathcal{R}(\mathbf{X}\varphi) = \mathbf{X}(run_i \mathbf{R}(st \vee \varphi))$

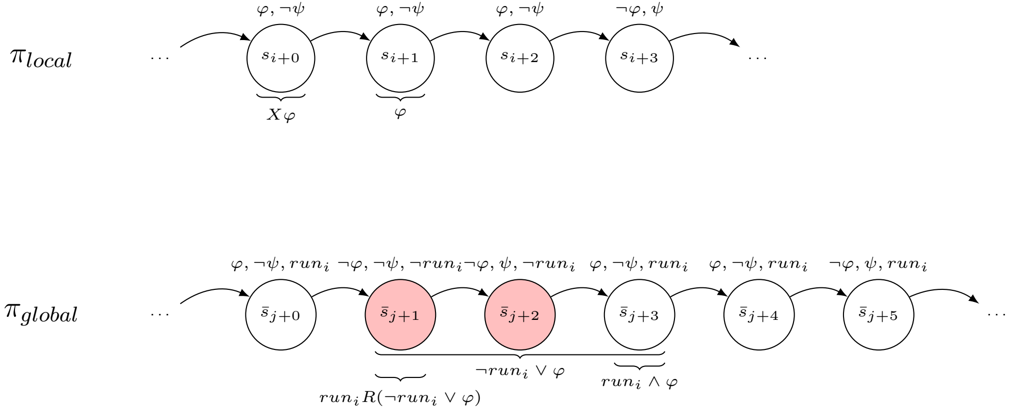




Example neXt

$Prop_{loc} = \mathbf{X}\varphi$

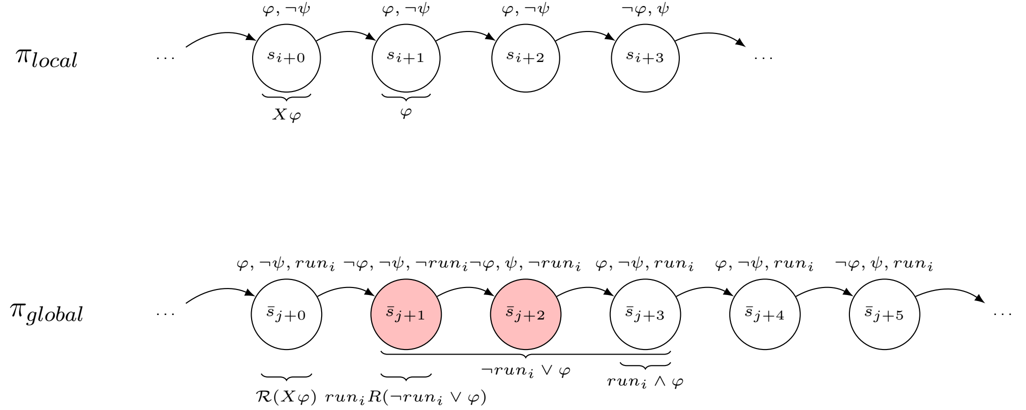
$Prop_{glob} = \mathcal{R}(\mathbf{X}\varphi) = \mathbf{X}(run_i\mathbf{R}(st \vee \varphi))$



Example neXt

$Prop_{loc} = \mathbf{X}\varphi$

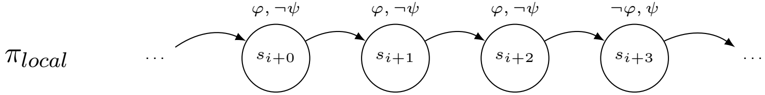
$Prop_{glob} = \mathcal{R}(\mathbf{X}\varphi) = \mathbf{X}(run_i \mathbf{R}(st \vee \varphi))$



Example Until

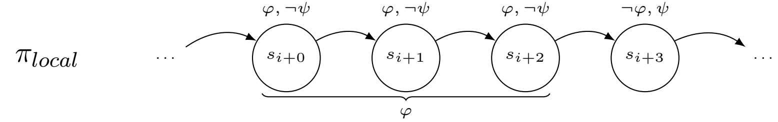
$Prop_{loc} = \varphi \mathbf{U} \psi$

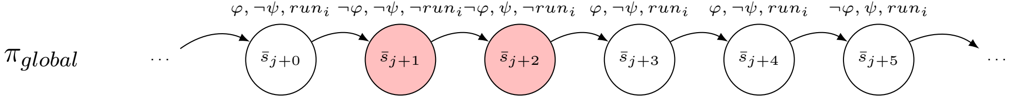
$Prop_{glob} = \mathcal{R}(\psi_1 \mathbf{U} \psi_2) = (\neg run_i \vee \psi_1) \mathbf{U} (run_i \wedge \psi_2)$



Example Until

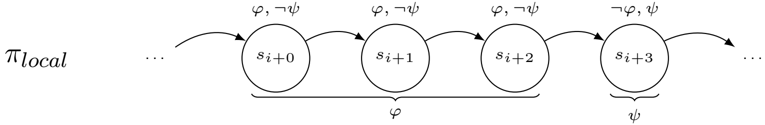
$Prop_{loc} = \varphi \mathbf{U} \psi$ $\qquad$ $Prop_{glob} = \mathcal{R}(\psi_1 \mathbf{U} \psi_2) = (\neg run_i \vee \psi_1)\mathbf{U}(run_i \wedge \psi_2)$

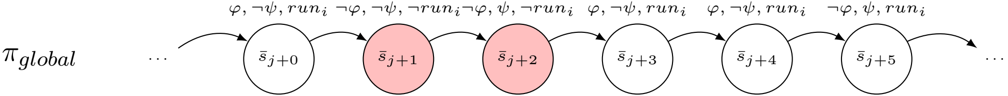




Example Until

$Prop_{loc} = \varphi \mathbf{U} \psi$ $\qquad$ $Prop_{glob} = \mathcal{R}(\psi_1 \mathbf{U} \psi_2) = (\neg run_i \vee \psi_1) \mathbf{U} (run_i \wedge \psi_2)$

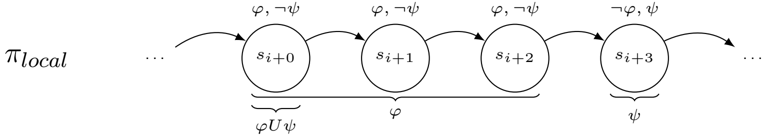


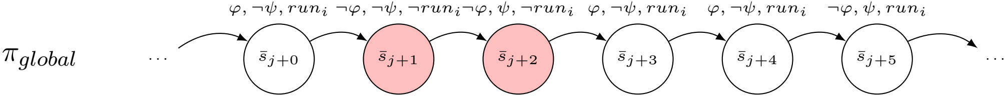


Example Until

$Prop_{loc} = \varphi \mathbf{U} \psi$

$Prop_{glob} = \mathcal{R}(\psi_1 \mathbf{U} \psi_2) = (\neg run_i \vee \psi_1)\mathbf{U}(run_i \wedge \psi_2)$

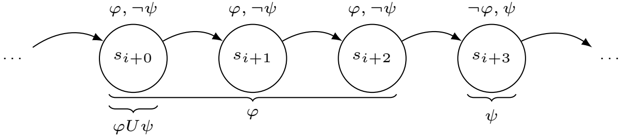




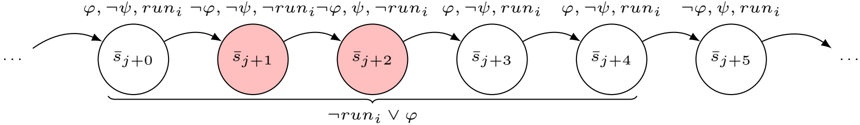
Example Until

$$Prop_{loc} = \varphi \mathbf{U} \psi \qquad Prop_{glob} = \mathcal{R}(\psi_1 \mathbf{U} \psi_2) = (\neg run_i \vee \psi_1)\mathbf{U}(run_i \wedge \psi_2)$$

π_{local}



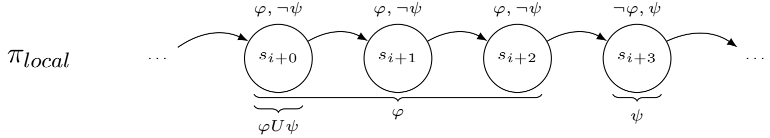
π_{global}

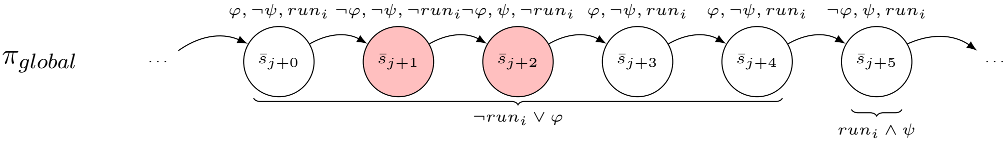


Example Until

$Prop_{loc} = \varphi \mathbf{U} \psi$

$Prop_{glob} = \mathcal{R}(\psi_1 \mathbf{U} \psi_2) = (\neg run_i \vee \psi_1) \mathbf{U} (run_i \wedge \psi_2)$

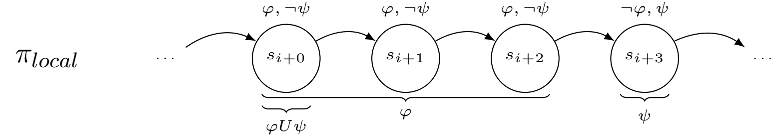


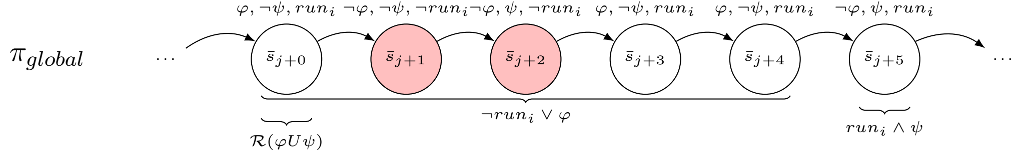


Example Until

$Prop_{loc} = \varphi \mathbf{U} \psi$

$Prop_{glob} = \mathcal{R}(\psi_1 \mathbf{U} \psi_2) = (\neg run_i \vee \psi_1)\mathbf{U}(run_i \wedge \psi_2)$





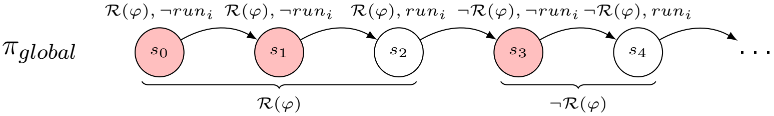
Optimization

Properties with input and outputs:

- Properties are over input and output variables
- output variables do not change when run_i does not holds

Properties with input and outputs:

- Properties are over input and output variables
- output variables do not change when run_i does not holds

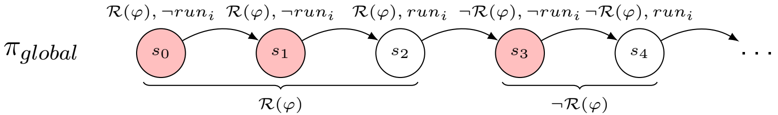


"Stutter tolerance" of φ:

Optimization

Properties with input and outputs:

- Properties are over input and output variables
- output variables do not change when run_i does not holds



"Stutter tolerance" of φ:

Applying stutter tolerance

- Stutter tolerant formulae are found syntactically (e.g. $\mathbf{U}, o_{var}$)
- If sub-formula is *"syntactically" stutter tolerant*, then it is not necessary to apply rewriting to the current op:
  - $\mathcal{R}^\theta(o1_{var}\mathbf{U}o2_{var}) = o1_{var}\mathbf{U}o2_{var}$
  - $\mathcal{R}^\theta(\mathbf{X}(o1_{var}\mathbf{U}o2_{var})) = \mathbf{X}(o1_{var}\mathbf{U}o2_{var})$
- Stutter tolerance also used for $\mathcal{R}^*$

Applying rewriting to contract refinement

Sync:

$$\text{Impl}: \bigwedge_{C_i \in Sub(C)} (A_i \to G_i) \to (A \to G)$$

$$\text{Env}: \text{For all } C_i \in Sub(C): \bigwedge_{C_j \in Sub(C) \setminus \{C_i\}} (A_j \to G_j) \to (A \to A_i)$$

Applying rewriting to contract refinement

Sync:

$$\textsf{Impl}: \bigwedge_{C_i \in Sub(C)} (A_i \to G_i) \to (A \to G)$$

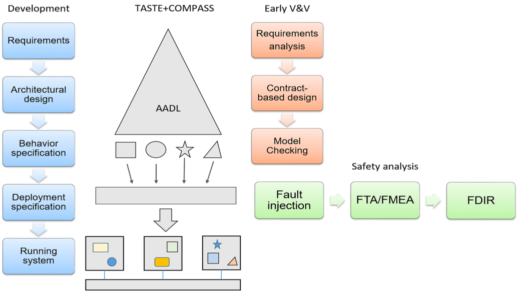
$$\textsf{Env}: \text{For all } C_i \in Sub(C): \bigwedge_{C_j \in Sub(C) \setminus \{C_i\}} (A_j \to G_j) \to (A \to A_i)$$

Async:

$$\textsf{Impl}: \bigwedge_{C_i \in Sub(C)} (\mathcal{R}_i^*(A_i) \to \mathcal{R}_i^*(G_i)) \wedge \alpha \to (A \to G)$$

$$\textsf{Env}: \forall_{C_i \in Sub(C)}: \bigwedge_{C_j \in Sub(C) \setminus \{C_i\}} (\mathcal{R}_j^*(A_j) \to \mathcal{R}_j^*(G_j)) \wedge \alpha \to (A \to \mathcal{R}_i^*(A_i))$$

COMPASTA[IMBSA22, CEAS23]



- HW + SW components composition
- Finite Communication buffers

EVA[TACAS23]

- Tool for compositional verification of AUTOSAR (automotive) systems
- Contract refinement using OCRA as backend
- Event based scheduling: $\mathbf{G}(change(BrakeSens1) \to \mathbf{X} RBrakeCmd.run)$
- Periodic scheduling: $\mathbf{F}_{\leq 1} CC.run \wedge \mathbf{G}(CC.run \to \mathbf{X}(\neg CC.run \mathbf{U}_{[}1,1] CC.run))$

Supporting finite scheduling of components

- We want to permit finite scheduling of components
- sub-components might crash or the scheduler might not be fair.

Supporting finite scheduling of components

- We want to permit finite scheduling of components
- sub-components might crash or the scheduler might not be fair.
- Composition locally possibly finite while system has infinite semantics.
- Local traces might have different length

$$\pi_1 := \bullet \to \bullet \to \bullet$$
$$\pi_2 := \bullet \to \bullet \to \bullet \to \bullet \to \bullet \to \dots$$
$$\pi_3 := \bullet \to \bullet$$

$$\pi = \pi_1 \times \pi_2 \times \pi_3$$

After end of local component, it stutters forever.

- Weak semantics (truncated LTL semantics of Eisner and FIsman)

Supporting finite scheduling of components

- We want to permit finite scheduling of components
- sub-components might crash or the scheduler might not be fair.
- Composition locally possibly finite while system has infinite semantics.
- Local traces might have different length

$$\pi_1 := \bullet \to \bullet \to \bullet$$

$$\pi_2 := \bullet \to \bullet \to \bullet \to \bullet \to \bullet \to \dots$$

$$\pi_3 := \bullet \to \bullet$$

$$\pi = \pi_1 \times \pi_2 \times \pi_3$$

After end of local component, it stutters forever.

- Weak semantics (truncated LTL semantics of Eisner and FIsman)

Weak semantics ($\models_-$):

- $\pi, i \models_- v \Leftrightarrow i \geq |\pi|$ or $\pi, i \models v \quad \pi, i \models_- \neg\varphi \Leftrightarrow \pi, i \not\models_+ \varphi$
- $\pi_f = \{a\}, \{b\} \models_- \mathbf{G}(a \to b)$ and $\pi_f \models_- \mathbf{G}(b \to \mathbf{X}\neg a)$
- Semantics identical to standard LTL for infinite traces

Supporting finite scheduling of components

- We want to permit finite scheduling of components
- sub-components might crash or the scheduler might not be fair.
- Composition locally possibly finite while system has infinite semantics.
- Local traces might have different length

$$\pi_1 := \bullet \rightarrow \bullet \rightarrow \bullet$$

$$\pi_2 := \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots$$

$$\pi_3 := \bullet \rightarrow \bullet$$

$$\pi = \pi_1 \times \pi_2 \times \pi_3$$

After end of local component, it stutters forever.

- Weak semantics (truncated LTL semantics of Eisner and FIsman)

Weak semantics ($\models_-$):

- $\pi, i \models_- v \Leftrightarrow i \geq |\pi|$ or $\pi, i \models v$ $\quad \pi, i \models_- \neg\varphi \Leftrightarrow \pi, i \not\models_+ \varphi$
- $\pi_f = \{a\}, \{b\} \models_- \mathbf{G}(a \rightarrow b)$ and $\pi_f \models_- \mathbf{G}(b \rightarrow \mathbf{X}\neg a)$
- Semantics identical to standard LTL for infinite traces

Relative safety

Liveness in verification, safety and relative safety

Problem:

- Liveness checking is hard!
- Our composition makes it impossible to reduce the problem to safety!

Idea:

- Try to not check the "liveness part" of the property
- Use relative safety[Henzinger91]. A property "becomes" safety when another property holds.
- $\alpha \to \varphi$ is relative safety to α if φ is safety

NOTE: invariant checking of $P \neq$ checking of $\mathbf{G}P$ (deadlocks and livelocks)

Properties classification

Safety property:
$P \subseteq \Sigma^\omega$ is a *safety property* iff $\forall \pi \in \Sigma^\omega$ s.t. $\pi \not\models P$, $\exists \pi_f \in Pref(\pi)$ s.t. $\forall \pi^\omega \in \Sigma^\omega : \pi_f \pi^\omega \not\models P$.

$$\underbrace{\bullet \rightarrow \bullet \rightarrow \bullet}_{\text{"bad prefix"}\,\pi_f} \rightarrow \bullet \rightarrow \bullet \rightarrow \cdots \in P$$

Properties classification

Safety property:
$P \subseteq \Sigma^\omega$ is a *safety property* iff $\forall \pi \in \Sigma^\omega$ s.t. $\pi \not\models P$, $\exists \pi_f \in Pref(\pi)$ s.t. $\forall \pi^\omega \in \Sigma^\omega : \pi_f \pi^\omega \not\models P$.

$$\underbrace{\bullet \rightarrow \bullet \rightarrow \bullet}_{\text{"bad prefix"}\,\pi_f} \rightarrow \bullet \rightarrow \bullet \rightarrow \cdots \in P$$

Liveness property:
$A \subseteq \Sigma^\omega$ is a *liveness property* iff $\forall \pi \in \Sigma^\omega, \forall \pi_f \in Pref(\pi)\ \exists \pi^\omega \in \Sigma^\omega : \pi_f \pi^\omega \models A$.

$$\underbrace{\ldots}_{\text{Any prefix}} \rightarrow \underbrace{\bullet \rightarrow \bullet \rightarrow \ldots}_{\pi^\omega} \in A$$

Properties classification

Safety property:
$P \subseteq \Sigma^\omega$ is a *safety property* iff $\forall \pi \in \Sigma^\omega$ s.t. $\pi \not\models P$, $\exists \pi_f \in Pref(\pi)$ s.t. $\forall \pi^\omega \in \Sigma^\omega : \pi_f \pi^\omega \not\models P$.

$$\underbrace{\bullet \to \bullet \to \bullet}_{\text{"bad prefix"}\,\pi_f} \to \bullet \to \bullet \to \cdots \in P$$

Liveness property:
$A \subseteq \Sigma^\omega$ is a *liveness property* iff $\forall \pi \in \Sigma^\omega, \forall \pi_f \in Pref(\pi)\ \exists \pi^\omega \in \Sigma^\omega : \pi_f \pi^\omega \models A$.

$$\underbrace{\ \cdots\ }_{\text{Any prefix}} \to \underbrace{\bullet \to \bullet \to \dots}_{\pi^\omega} \in A$$

SafetyLTL: $\varphi := a \mid \neg a \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{R} \varphi$ $\qquad safetyLTL = safety \cap LTL$

Relative safety

Let P and A be two properties. P is safety relative to A iff

$$\forall \pi \in A \text{ s.t. } \pi \notin P, \exists \pi_f \in Pref(\pi) \text{ s.t.}$$
$$\forall \pi^\omega \in \Sigma^\omega : \text{ if } \pi_f \pi^\omega \in A \text{ then } \pi_f \pi^\omega \notin P$$

Notable examples:

- φ_S is safety relative to $\top$.
- $\mathbf{G}p \rightarrow \mathbf{G}q$ is safety relative to $\mathbf{G}p$.
- φ_{SafMTL} is safety relative to non-zenoness.
- $\varphi \mathbf{U} \psi$ is safety relative to $\mathbf{F}\psi$

Related works

Liveness to safety

- Encodes absense of lasso-shaped path with an invariant
- Copy all the variables

A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. In International Workshop on Formal Methods for Industrial Critical Systems, 2002

K-liveness

- Introduce a counter c that counts the occurrence of a fairness condition
- Checks **FG**$\neg f$ by invariant checking $c \leq k$
- Can only prove valid properties

K. Claessen and N. Sörensson. A liveness checking algorithm that counts. 2012 Formal Methods in Computer-Aided Design (FMCAD), pages 52–59, 2012.

Basic algorithm

Check $\underbrace{\mathcal{M}}_{SymbolicTransitionSystem} \models \alpha \rightarrow \varphi$

$$\boxed{\mathcal{M}' := \mathcal{M} \times \mathcal{M}_{\alpha_S}}$$

- $\alpha := \alpha_S \wedge \alpha_L$
- φ safetyLTL

Basic algorithm

Check $\underbrace{\mathcal{M}}_{SymbolicTransitionSystem} \models \alpha \rightarrow \varphi$

$\mathcal{M}' := \mathcal{M} \times \mathcal{M}_{\alpha_S}$

$\mathcal{M}' \models_{INVAR} \varphi$

- $\alpha := \alpha_S \wedge \alpha_L$
- φ safetyLTL

Check $\underbrace{\mathcal{M}}_{SymbolicTransitionSystem} \models \alpha \rightarrow \varphi$

$\mathcal{M}' := \mathcal{M} \times \mathcal{M}_{\alpha_S}$

$\mathcal{M}' \models_{INVAR} \varphi$

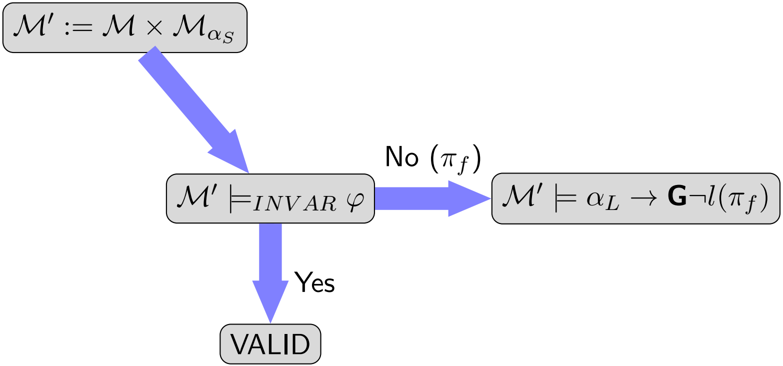
Yes

VALID

- $\alpha := \alpha_S \wedge \alpha_L$
- φ safetyLTL
- $\mathcal{M}' \models_{INVAR} \varphi \Rightarrow \mathcal{M}' \models \varphi$ ($\models_{INVAR}$ reduces safetyLTL to invariant)

Basic algorithm

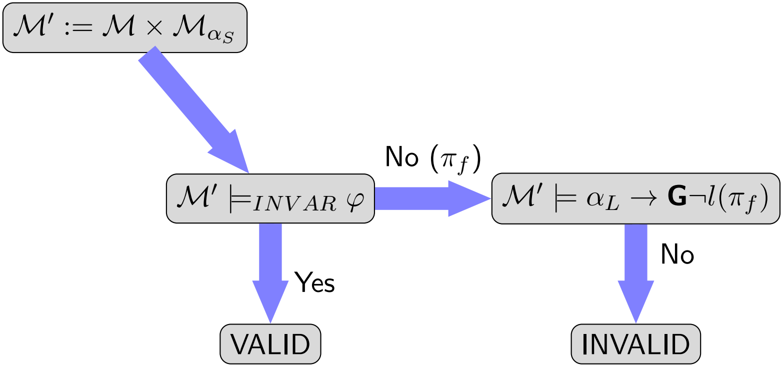
Check $\underbrace{\mathcal{M}}_{SymbolicTransitionSystem} \models \alpha \rightarrow \varphi$



- $\alpha := \alpha_S \wedge \alpha_L$
- φ safetyLTL
- $\mathcal{M}' \models_{INVAR} \varphi \Rightarrow \mathcal{M}' \models \varphi$ ($\models_{INVAR}$ reduces safetyLTL to invariant)
- Try to extend the trace to infinity

Basic algorithm

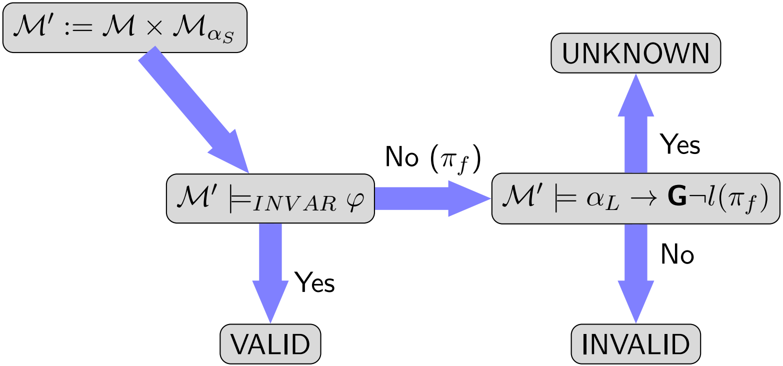
Check $\underbrace{\mathcal{M}}_{SymbolicTransitionSystem} \models \alpha \rightarrow \varphi$



- $\alpha := \alpha_S \wedge \alpha_L$
- φ safetyLTL
- $\mathcal{M}' \models_{INVAR} \varphi \Rightarrow \mathcal{M}' \models \varphi$ ($\models_{INVAR}$ reduces safetyLTL to invariant)
- Try to extend the trace to infinity

Basic algorithm

Check $\underbrace{\mathcal{M}}_{SymbolicTransitionSystem} \models \alpha \rightarrow \varphi$



- $\alpha := \alpha_S \wedge \alpha_L$
- φ safetyLTL
- $\mathcal{M}' \models_{INVAR} \varphi \Rightarrow \mathcal{M}' \models \varphi$ ($\models_{INVAR}$ reduces safetyLTL to invariant)
- Try to extend the trace to infinity

CEGAR Algorithm

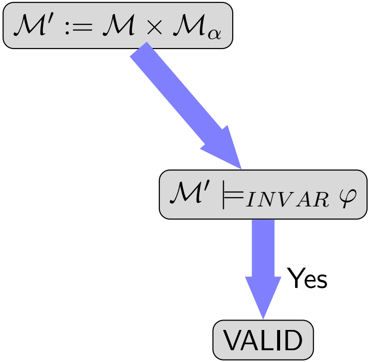
$$\boxed{\mathcal{M}' := \mathcal{M} \times \mathcal{M}_\alpha}$$

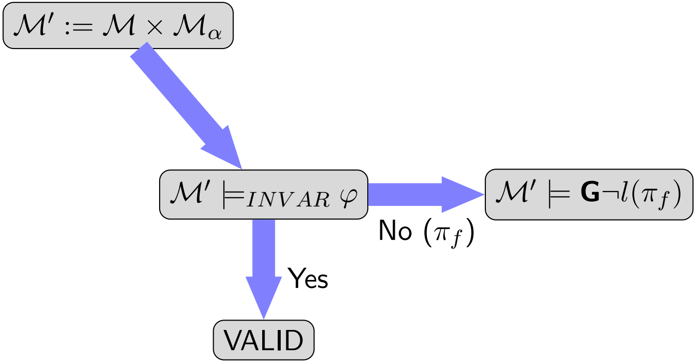
CEGAR Algorithm

$$\mathcal{M}' := \mathcal{M} \times \mathcal{M}_\alpha$$

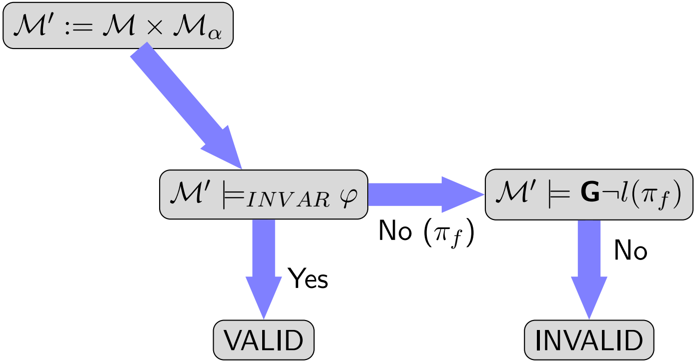
$$\mathcal{M}' \models_{INVAR} \varphi$$

CEGAR Algorithm

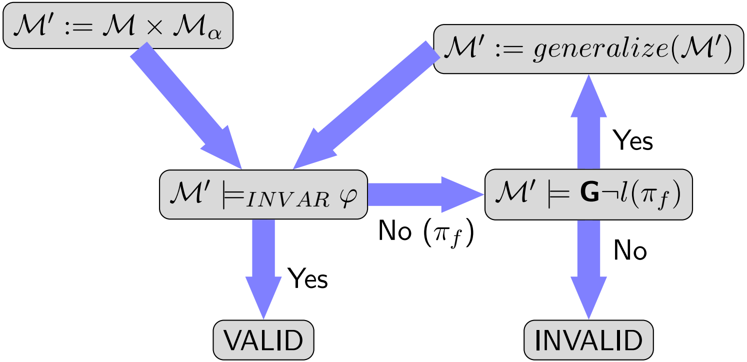



$\mathcal{M}' := \mathcal{M} \times \mathcal{M}_\alpha$
$\mathcal{M}' \models_{INVAR} \varphi$
No (π_f)
$\mathcal{M}' \models \mathbf{G}\neg l(\pi_f)$
Yes
VALID

CEGAR Algorithm



CEGAR Algorithm



- $generalize(\mathcal{M}', l(\pi_f))$ block unfair states
- If $\mathcal{M}$ is a finite state STS, the algorithm is complete

LTL to Symbolic Transition System ($\mathcal{M}_{\neg\phi}$)

- $V_{\neg\phi} = V \cup \{v_{X\beta} \mid \mathbf{X}\beta \in Sub(\phi)\} \cup \{v_{\mathbf{X}(\beta_1 \mathbf{U} \beta_2)} \mid \beta_1 \mathbf{U} \beta_2 \in Sub(\phi)\} \cup \{v_{\mathbf{Y}\beta} \mid \mathbf{Y}\beta \in Sub(\phi)\} \cup \{v_{\mathbf{Y}\beta_1 \mathbf{S} \beta_2} \mid \beta_1 \mathbf{S} \beta_2 \in Sub(\phi)\}$
- $I_{\neg\phi} = Enc(\neg\phi) \wedge \bigwedge_{v_{\mathbf{Y}\beta} \in V_{\neg\phi}} \neg v_{\mathbf{Y}\beta}$
- $T_{\neg\phi} = \bigwedge_{v_{\mathbf{X}\beta} \in V_{\neg\phi}} v_{\mathbf{X}\beta} \leftrightarrow Enc(\beta)' \wedge \bigwedge_{v_{\mathbf{Y}\beta} \in V_{\neg\phi}} Enc(\beta) \leftrightarrow v'_{\mathbf{Y}\beta}$
- $F_{\neg\phi} = \{Enc(\beta_1 \mathbf{U} \beta_2) \rightarrow \beta_2 \mid \beta_1 \mathbf{U} \beta_2 \in Sub(\phi)\}$

where Sub is a function that maps a formula ϕ to the set of its subformulas, and Enc is defined recursively as:

- $Enc(\top) = \top$
- $Enc(v) = v$
- $Enc(\phi_1 \wedge \phi_2) = Enc(\phi_1) \wedge Enc(\phi_2)$
- $Enc(\neg\phi_1) = \neg Enc(\phi_1)$
- $Enc(\mathbf{X}\phi_1) = v_{\mathbf{X}\phi_1}$
- $Enc(\phi_1 \mathbf{U} \phi_2) = Enc(\phi_2) \vee (Enc(\phi_1) \wedge v_{\mathbf{X}(\phi_1 \mathbf{U} \phi_2)})$
- $Enc(\mathbf{Y}\phi_1) = v_{\mathbf{Y}\phi_1}$
- $Enc(\phi_1 \mathbf{S} \phi_2) = Enc(\phi_2) \vee (Enc(\phi_1) \wedge v_{\mathbf{Y}(\phi_1 \mathbf{S} \phi_2)})$

SafetyLTL to Symbolic Transition System

High level idea:

- Rewrite $\neg\phi$ in nnf
- Construct STS of $\neg\phi$ (similar to LTL2SMV)
- Compute invariant $INV_\phi := \neg(\bigwedge_{v_{\mathbf{X}\beta}} \neg v_{\mathbf{X}\beta})$

SafetyLTL to Symbolic Transition System

High level idea:

- Rewrite $\neg\phi$ in nnf
- Construct STS of $\neg\phi$ (similar to LTL2SMV)
- Compute invariant $INV_\phi := \neg(\bigwedge_{v_{\mathbf{X}\beta}} \neg v_{\mathbf{X}\beta})$

$$\mathcal{M}_{\neg\phi} := \langle \mathcal{V}_{\neg\phi}, \mathcal{I}_{\neg\phi}, \mathcal{T}_{\neg\phi} \rangle$$

$$I_{\neg\phi} = Enc(\neg\phi) \wedge \bigwedge_{v_{\mathbf{Y}\beta} \in V_{\neg\phi}} \neg v_{\mathbf{Y}\beta}$$

$$T_{\neg\phi} = \bigwedge_{v_{\mathbf{X}\beta} \in V_{\neg\phi}} v_{\mathbf{X}\beta} \rightarrow Enc(\beta)') \wedge$$

$$\bigwedge_{v_{\mathbf{Y}\beta} \in V_{\neg\phi}} v'_{\mathbf{Y}\beta} \rightarrow Enc(\beta))$$

SafetyLTL to Symbolic Transition System

$$\mathcal{M}_{\neg\phi} := \langle \mathcal{V}_{\neg\phi}, \mathcal{I}_{\neg\phi}, \mathcal{T}_{\neg\phi} \rangle$$

High level idea:

- Rewrite $\neg\phi$ in nnf
- Construct STS of $\neg\phi$ (similar to LTL2SMV)
- Compute invariant $INV_\phi := \neg(\bigwedge_{v_{\mathbf{X}\beta}} \neg v_{\mathbf{X}\beta})$

$$I_{\neg\phi} = Enc(\neg\phi) \wedge \bigwedge_{v_{\mathbf{Y}\beta} \in V_{\neg\phi}} \neg v_{\mathbf{Y}\beta}$$

$$T_{\neg\phi} = \bigwedge_{v_{\mathbf{X}\beta} \in V_{\neg\phi}} v_{\mathbf{X}\beta} \rightarrow Enc(\beta)')\wedge$$

$$\bigwedge_{v_{\mathbf{Y}\beta} \in V_{\neg\phi}} v'_{\mathbf{Y}\beta} \rightarrow Enc(\beta)$$

$Enc(\varphi)$:

- $Enc(\phi_1 \wedge \phi_2) = Enc(\phi_1) \wedge Enc(\phi_2)$, $Enc(\phi_1 \vee \phi_2) = Enc(\phi_1) \vee Enc(\phi_2)$
- $Enc(\neg\phi_1) = \neg Enc(\phi_1)$
- $Enc(\mathbf{X}\phi_1) = v_{\mathbf{X}\phi_1}$
- $Enc(\phi_1 \mathbf{U} \phi_2) = Enc(\phi_2) \vee (Enc(\phi_1) \wedge v_{\mathbf{X}(\phi_1 \mathbf{U} \phi_2)})$

Extending invariant check with lookahead

- If $\mathcal{M}'$ has *livelocks*, multiple steps are required
- We try to take longer counterexamples to rule out deadlock states ($\approx AX^n\top$)

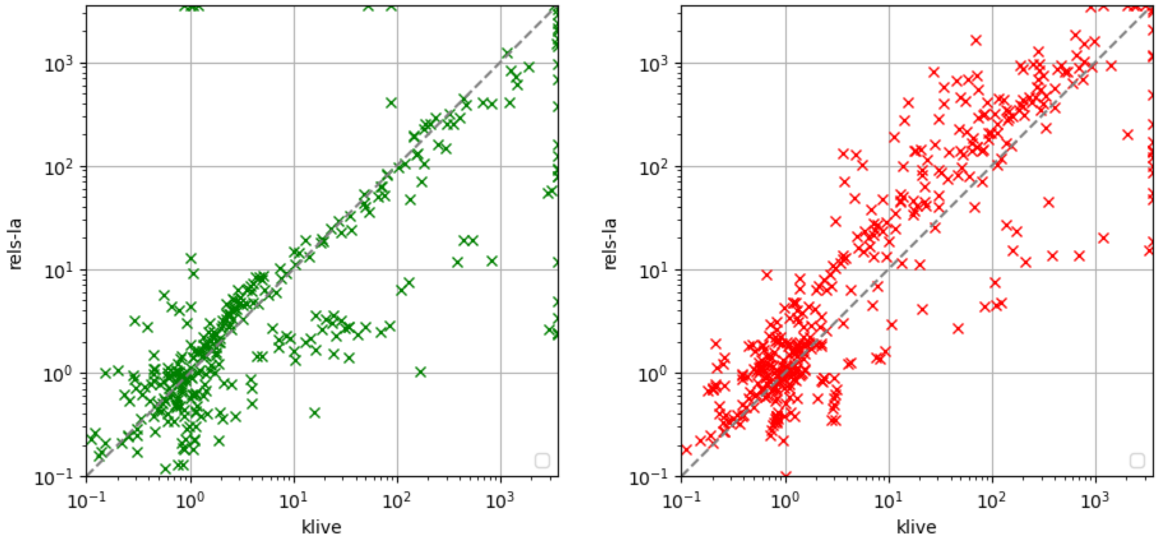
$$safetyLTL2STSLa(\mathcal{M}', \varphi, n) :=$$

1. $\langle \mathcal{M}_{\neg\phi}^{saf}, INV_\varphi \rangle := safetyLTLSTS(\varphi)$
2. $\mathcal{M}_{\neg\phi}^{saf} \leftarrow \langle \mathcal{V}' \cup \{la\}, \mathcal{I}' \wedge la = 0,$
   $\mathcal{T}' \wedge (la > 0 \vee \neg INV_\varphi \rightarrow la' = la + 1) \wedge (la = 0 \wedge INV_\varphi \rightarrow la' = 0)) \rangle$
3. $Return(\langle \mathcal{M}_{\neg\phi}^{saf}, la < n \rangle)$

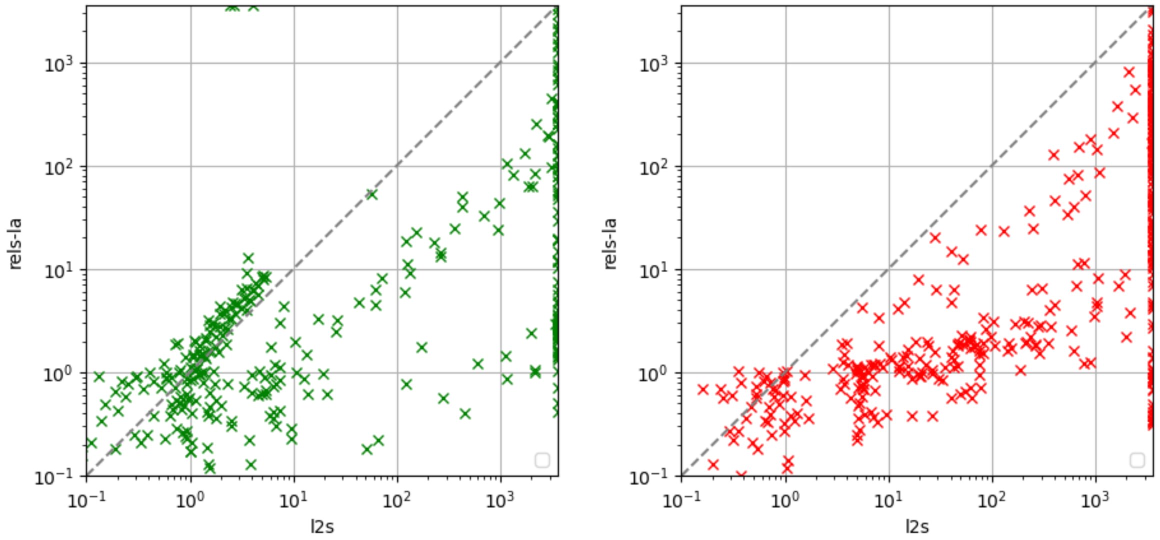
Experimental evaluation

- Comparison with k-liveness, liveness to safety.
- A/G contracts (e.g. Wheel Brake System)
- Bounded response
- Asynchronous composition
- NuSMV models
- Monitor-Sensor

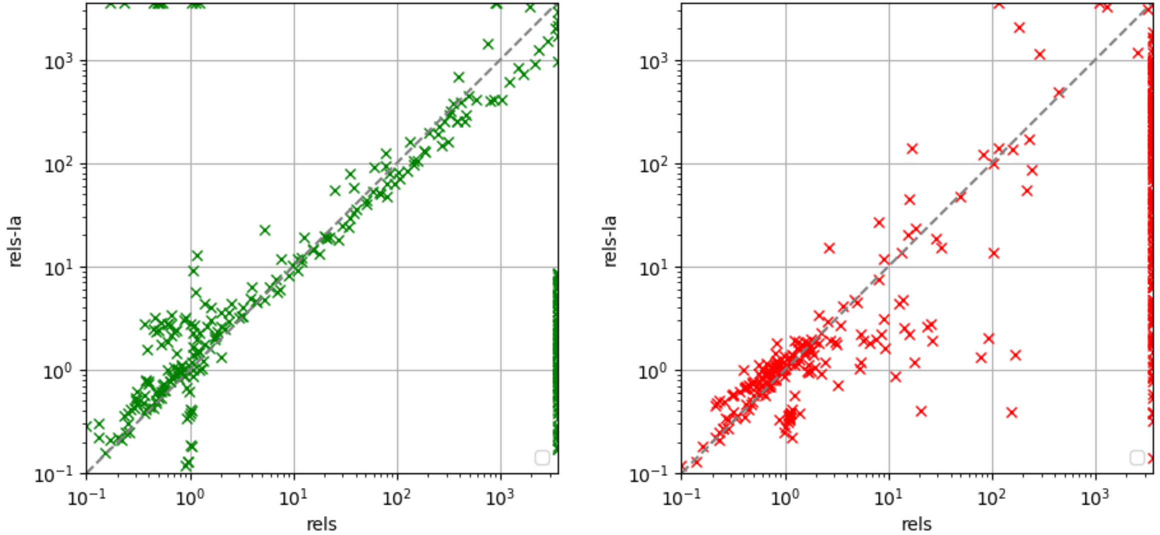
Comparison with k-liveness



Comparison with liveness to safety



Comparison with rels-no-la



Open problems and possible extensions

- Performance strongly depends on the translation from property to STS (The construction might introduce deadlocks)
- $\alpha \rightarrow \varphi$ is a restricted application of relative safety
- Generalization is still a work in progress (based on inductive invariant extraction from k-liveness)
- Possible direction would be to specialise this for contract refinement