Predictive Runtime Enforcement

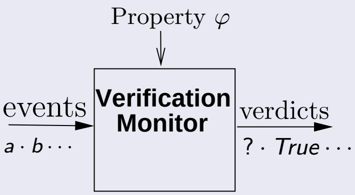
Srinivas Pinisetty[1], Viorel Preoteasa[1], Stavros Tripakis[1,2], Thierry Jéron[3], Yliès Falcone[4], Hervé Marchand[3]

Aalto University, Finland
University of California, Berkeley
INRIA Rennes - Bretagne Atlantique, France
LIG, Université Grenoble, INRIA, Grenoble, France

Computational Logic Day 2016, Aalto University, Finland

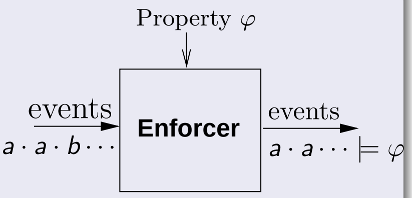
Runtime verification and enforcement

Runtime verification



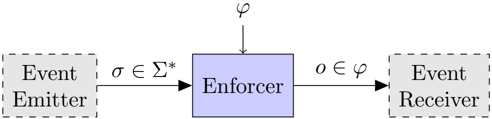
- Does σ satisfy φ ?
- Output: stream of **verdicts**.

Runtime enforcement

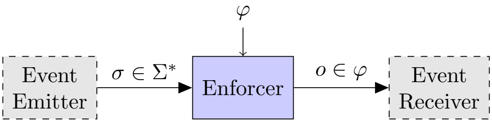


- Input: stream of events.
- Modified to satisfy the property.
- Output: stream of **events**.

Runtime enforcement (previous work: **non-predictive**)



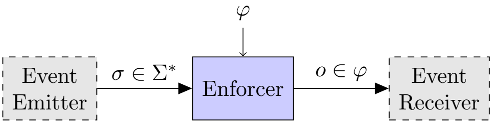
Runtime enforcement (previous work: **non-predictive**)



Enforcer for φ operating at runtime

- φ: any regular property (defined as automaton).

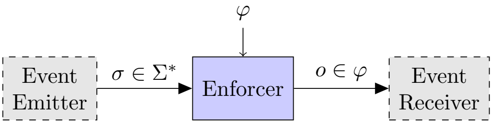
Runtime enforcement (previous work: **non-predictive**)



Enforcer for φ operating at runtime

- φ: any regular property (defined as automaton).
- An enforcer behaves as a function $E : \Sigma^* \rightarrow \Sigma^*$.
  - Input ($\sigma \in \Sigma^*$): any sequence of events over Σ (Event emitter is a black-box).

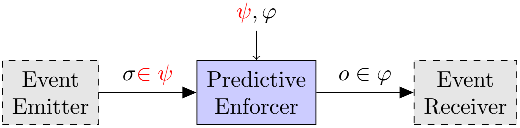
Runtime enforcement (previous work: **non-predictive**)



Enforcer for φ operating at runtime

- φ: any regular property (defined as automaton).
- An enforcer behaves as a function $E : \Sigma^* \rightarrow \Sigma^*$.
  - Input ($\sigma \in \Sigma^*$): any sequence of events over Σ (Event emitter is a black-box).
  - Output ($o \in \Sigma^*$): a sequence of events such that $o \models \varphi$.

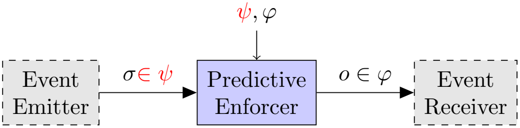
Predictive runtime enforcement problem (this work)



Predictive enforcer for φ operating at runtime

- Given property φ (to enforce) and input property ψ defined as automaton.
- Automatically synthesize an enforcer.

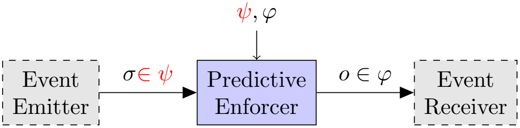
Predictive runtime enforcement problem (this work)



Predictive enforcer for φ operating at runtime

- Given property φ (to enforce) and input property ψ defined as automaton.
- Automatically synthesize an enforcer.
  - Input ($\sigma \in \psi$): Event emitter is not a black-box.
  - Output ($o \in \Sigma^*$): a sequence of events $o \models \varphi$.

Predictive runtime enforcement problem (this work)



Predictive enforcer for φ operating at runtime

- Given property φ (to enforce) and input property ψ defined as automaton.
- Automatically synthesize an enforcer.
  - Input ($\sigma \in \psi$): Event emitter is not a black-box.
  - Output ($o \in \Sigma^*$): a sequence of events $o \models \varphi$.
  - Predictive enforcer should satisfy soundness, transparency, monotonicity, and urgency constraints.
  - Urgency related to using ψ and release input events earlier whenever possible.

Predictive runtime enforcement (motivations)

Motivations

- Consider a-priori knowledge of the system (event emitter) is available.
  - Model, information extracted using static-analysis etc.
- Provide a-priori knowledge of the system (event emitter) to the enforcer.
  - Event emitter is not a black-box.
  - How enforcer can benefit from model/knowledge of the system?
  - Does it help to provide better QoS (eg: output some events earlier)?

Example

- Non-safety properties (release events earlier instead of delaying).

Related works

Runtime Enforcement: non-predictive

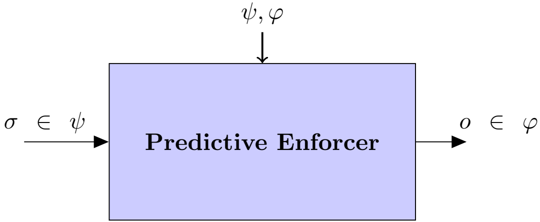
- Enforceable security policies – F. B. Schneider et al-2000.
- Runtime enforcement of non-safety policies – J. Ligatti et al-2009.
- Enforcement monitoring wrt. the safety-progress classification of properties – Y. Falcone et al-2010.
- Runtime enforcement of timed properties – S. Pinisetty et al-2012.
- Runtime enforcement for reactive systems – R. Bloem et al-2015.

Outline

Outline

Predictive enforcer

Given properties ψ (input property) and φ (to enforce):



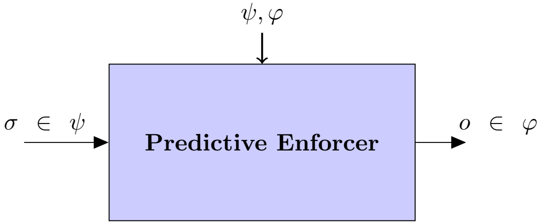
What can an enforcer do?

Enforcer augmented with a **memorization mechanism**.

- CAN delay events.
- CANNOT insert nor delete events.
- CANNOT change the order of events.

Formal problem definition

Properties ψ (input property) and φ (to enforce).



Predictive enforcer for ψ, φ

Given properties $\psi, \varphi \subseteq \Sigma^*$, a *predictive enforcer* is a function $E_{\psi,\varphi} : \Sigma^* \to \Sigma^*$ satisfying the following constraints:

1. Soundness
2. Transparency
3. Monotonicity
4. Urgency

Soundness

Output is correct (satisfies φ)

$$\forall \sigma \in \psi : E_{\psi,\varphi}(\sigma) \neq \epsilon \implies E_{\psi,\varphi}(\sigma) \in \varphi.$$

Transparency

TR1: events can be delayed

$$\forall \sigma \in \Sigma^* : E_{\psi,\varphi}(\sigma) \preccurlyeq \sigma.$$

Transparency

TR1: events can be delayed

$$\forall \sigma \in \Sigma^* : E_{\psi,\varphi}(\sigma) \preccurlyeq \sigma.$$

TR2: observed input satisfies φ

$$\forall \sigma \in \Sigma^* : \sigma \in \varphi \implies E_{\psi,\varphi}(\sigma) = \sigma$$

Monotonicity

Modify output only by appending new events

$$\forall \sigma, \sigma' \in \Sigma^* : \sigma \preccurlyeq \sigma' \implies E_{\psi,\varphi}(\sigma) \preccurlyeq E_{\psi,\varphi}(\sigma')$$

Urgency

Release observed input ASAP predicting future input using ψ

$$\forall \sigma \in \Sigma^* : (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \wedge \sigma \cdot \sigma' \in \varphi) \implies E_{\psi,\varphi}(\sigma) = \sigma$$

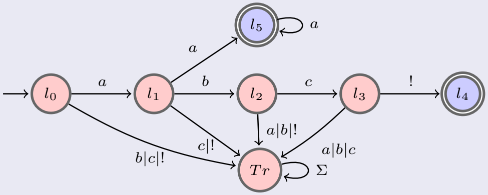
Urgency

Release observed input ASAP predicting future input using ψ

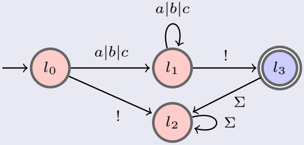
$$\forall \sigma \in \Sigma^* : (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \wedge \sigma \cdot \sigma' \in \varphi) \implies E_{\psi,\varphi}(\sigma) = \sigma$$

Example

Input property ψ



Property to enforce φ



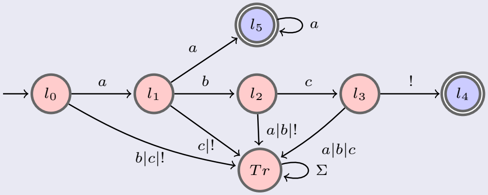
Urgency

Release observed input ASAP predicting future input using ψ

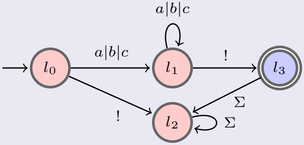
$$\forall \sigma \in \Sigma^* : (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \wedge \sigma \cdot \sigma' \in \varphi) \implies E_{\psi,\varphi}(\sigma) = \sigma$$

Example

Input property ψ



Property to enforce φ



- $\sigma = a \notin \varphi$

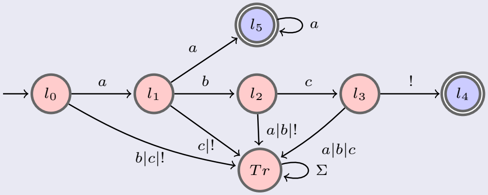
Urgency

Release observed input ASAP predicting future input using ψ

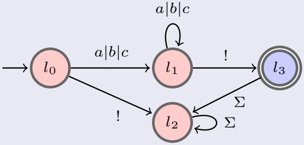
$$\forall \sigma \in \Sigma^* : (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \wedge \sigma \cdot \sigma' \in \varphi) \implies E_{\psi,\varphi}(\sigma) = \sigma$$

Example

Input property ψ



Property to enforce φ



- $\sigma = a \notin \varphi$, $\sigma_{con} = \{b \cdot c \cdot !, a \cdot a \cdot a \cdots a\}$

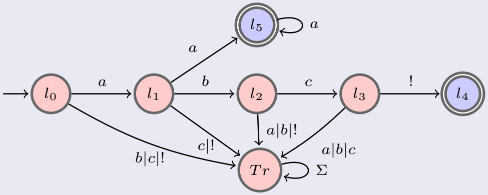
Urgency

Release observed input ASAP predicting future input using ψ

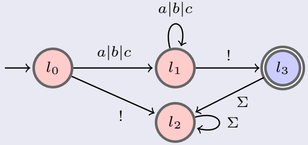
$$\forall \sigma \in \Sigma^* : (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \wedge \sigma \cdot \sigma' \in \varphi) \implies E_{\psi,\varphi}(\sigma) = \sigma$$

Example

Input property ψ



Property to enforce φ



- $\sigma = a \notin \varphi$, $\sigma_{con} = \{b \cdot c \cdot !, a \cdot a \cdot a \cdots a\}$, *WAIT*.

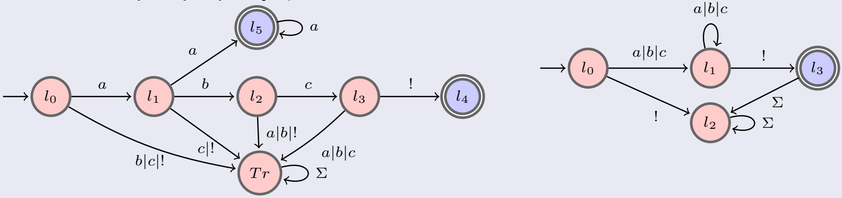
Urgency

Release observed input ASAP predicting future input using ψ

$$\forall \sigma \in \Sigma^* : (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \land \sigma \cdot \sigma' \in \varphi) \implies E_{\psi,\varphi}(\sigma) = \sigma$$

Example

Input property ψ Property to enforce φ



- $\sigma = a \notin \varphi$, $\sigma_{con} = \{b \cdot c \cdot !, a \cdot a \cdot a \cdots a\}$, *WAIT*.
- $\sigma = a \cdot \mathbf{b} \notin \varphi$

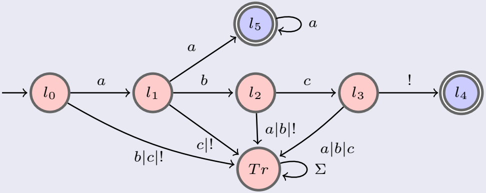
Urgency

Release observed input ASAP predicting future input using ψ

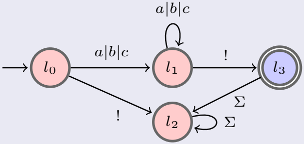
$$\forall \sigma \in \Sigma^* : (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \wedge \sigma \cdot \sigma' \in \varphi) \implies E_{\psi,\varphi}(\sigma) = \sigma$$

Example

Input property ψ



Property to enforce φ



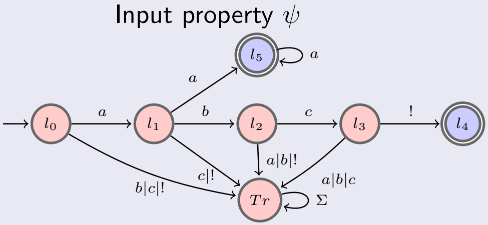
- $\sigma = a \notin \varphi$, $\sigma_{con} = \{b \cdot c \cdot !, a \cdot a \cdot a \cdots a\}$, *WAIT*.
- $\sigma = a \cdot \mathbf{b} \notin \varphi$, $\sigma_{con} = \{c \cdot !\}$

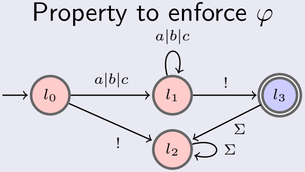
Urgency

Release observed input ASAP predicting future input using ψ

$$\forall \sigma \in \Sigma^* : (\forall \sigma_{\text{con}} \in \Sigma^* : \sigma \cdot \sigma_{\text{con}} \in \psi \implies \exists \sigma' \in \Sigma^* : \sigma' \preccurlyeq \sigma_{\text{con}} \wedge \sigma \cdot \sigma' \in \varphi) \implies E_{\psi,\varphi}(\sigma) = \sigma$$

Example





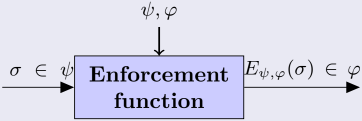
- $\sigma = a \notin \varphi$, $\sigma_{con} = \{b \cdot c \cdot !, a \cdot a \cdot a \cdots a\}$, *WAIT*.
- $\sigma = a \cdot \mathbf{b} \notin \varphi$, $\sigma_{con} = \{c \cdot !\}$, *RELEASE* $a \cdot b$.

Outline

Functional definition

$E_{\psi,\varphi} : \Sigma^* \to \Sigma^*$

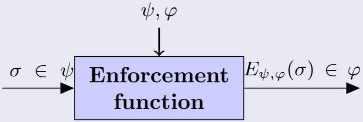
$$E_{\psi,\varphi}(\sigma) = \Pi_1\big(\mathsf{store}(\sigma)\big).$$



Functional definition

$E_{\psi,\varphi} : \Sigma^* \to \Sigma^*$

$$E_{\psi,\varphi}(\sigma) = \Pi_1\big(\text{store}(\sigma)\big).$$



$\text{store}_{\psi,\varphi} : \Sigma^* \to \Sigma^* \times \Sigma^*$

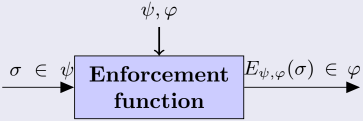
$\text{store}_{\psi,\varphi}(\sigma)$ is a pair:

1. output of the enforcer (prefix of σ),
2. suffix of σ which the enforcer cannot output yet.

Functional definition

$E_{\psi,\varphi} : \Sigma^* \to \Sigma^*$

$$E_{\psi,\varphi}(\sigma) = \Pi_1\big(\mathsf{store}(\sigma)\big).$$



$\mathrm{store}_{\psi,\varphi} : \Sigma^* \to \Sigma^* \times \Sigma^*$

$\mathrm{store}_{\psi,\varphi}(\sigma)$ is a pair:

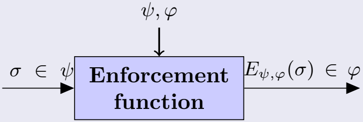
1. output of the enforcer (prefix of σ),
2. suffix of σ which the enforcer cannot output yet.

Suppose $(\sigma_s, \sigma_c) = \mathrm{store}_{\psi,\varphi}(\sigma)$

Functional definition

$E_{\psi,\varphi} : \Sigma^* \to \Sigma^*$

$$E_{\psi,\varphi}(\sigma) = \Pi_1\big(\text{store}(\sigma)\big).$$



$\text{store}_{\psi,\varphi} : \Sigma^* \to \Sigma^* \times \Sigma^*$

$\text{store}_{\psi,\varphi}(\sigma)$ is a pair:

1. output of the enforcer (prefix of σ),
2. suffix of σ which the enforcer cannot output yet.

Suppose $(\sigma_s, \sigma_c) = \text{store}_{\psi,\varphi}(\sigma)$

$$\text{store}_{\psi,\varphi}(\sigma \cdot a) = \begin{cases} (\sigma_s \cdot \sigma_c \cdot a, \epsilon) & \text{if } \kappa_{\psi,\varphi}(\sigma_s \cdot \sigma_c \cdot a), \\ (\sigma_s, \sigma_c \cdot a) & \text{otherwise} \end{cases}$$

where $\kappa_{\psi,\varphi}$ tests the hypothesis of the Urgency constraint.

Functional definition satisfies constraints

Theorem

The functional definition we previously saw satisfies the following constraints:

1. *Soundness*
2. *Transparency*
3. *Monotonicity*
4. *Urgency*

Isabelle proofs

https://github.com/isabelle-theory/PredictiveRuntimeEnforcement

Outline

Enforcement algorithm

Input: $\mathcal{A}_\psi = (Q_\psi, q_\psi, \Sigma, \delta_\psi, F_\psi)$, $\mathcal{A}_\varphi = (Q_\varphi, q_\varphi, \Sigma, \delta_\varphi, F_\varphi)$.

Online algorithm

initialize memory, automata current states;
while *True* **do**
    WAIT for input event;
    UPDATE current states;
    if $(\kappa_{\psi,\varphi}())$ **then**
        RELEASE memory content and input event;
    else
        ADD input event to memory;
    end
end

Implementation of $\kappa_{\psi,\varphi}$

Input: $\mathcal{A}_\psi = (Q_\psi, q_\psi, \Sigma, \delta_\psi, F_\psi)$, $\mathcal{A}_\varphi = (Q_\varphi, q_\varphi, \Sigma, \delta_\varphi, F_\varphi)$.

Testing $\kappa_{\psi,\varphi}(\sigma)$ by checking emptiness of a regular language

- $p =$ state in $\mathcal{A}_\psi$ upon reading σ.
- $q=$ state in $\mathcal{A}_\varphi$ upon reading σ.

$$\kappa_{\psi,\varphi}(\sigma) \iff \mathcal{L}(\mathcal{A}_\psi \times \overline{\mathcal{B}_\varphi}, (p,q)) = \emptyset$$

Implementation of $\kappa_{\psi,\varphi}$

Input: $\mathcal{A}_\psi = (Q_\psi, q_\psi, \Sigma, \delta_\psi, F_\psi)$, $\mathcal{A}_\varphi = (Q_\varphi, q_\varphi, \Sigma, \delta_\varphi, F_\varphi)$.

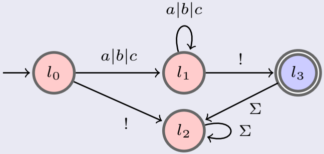
Testing $\kappa_{\psi,\varphi}(\sigma)$ by checking emptiness of a regular language

- p = state in $\mathcal{A}_\psi$ upon reading σ.
- q= state in $\mathcal{A}_\varphi$ upon reading σ.

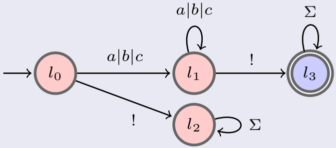
$$\kappa_{\psi,\varphi}(\sigma) \iff \mathcal{L}(\mathcal{A}_\psi \times \overline{\mathcal{B}_\varphi}, (p, q)) = \emptyset$$

Automaton B_φ

Automaton A_φ.



Automaton B_φ based on A_φ.



Enforcement algorithm

Predictive Enforcer

```
σ_c ← ϵ
p, q ← q_ψ, q_φ
C ← A_ψ × B̄_φ
while true do
    a ← await_event()
    p, q ← δ_ψ(p, a), δ_φ(q, a)
    if L(C, (p, q)) = ∅ then
        release(σ_c · a)
        σ_c ← ϵ
    else
        σ_c ← σ_c · a
    end if
end while
```

Complexity

- Product automaton $\mathcal{C}$, emptiness test for every state in $\mathcal{C}$ (off-line).
- Constant on-line time complexity.

Outline

Conclusion and Future Work

Conclusion

- Introduced predictive RE framework.
  - A-priori knowledge of system behavior ψ.
  - Predicting future using ψ (Urgency constraint).
  - Urgency ensures that enforcer reacts ASAP.

Conclusion and Future Work

Conclusion

- Introduced predictive RE framework.
  - A-priori knowledge of system behavior ψ.
  - Predicting future using ψ (Urgency constraint).
  - Urgency ensures that enforcer reacts ASAP.
- Properties φ and ψ are regular languages (modeled as automata).
- Enforcer synthesis.
- Algorithms implementing these mechanisms in constant on-line time.
- Proofs in Isabelle.
- Implementation: `https://github.com/SrinivasPinisetty/PredictiveRE`.

Conclusion and Future Work

Conclusion

- Introduced predictive RE framework.
  - A-priori knowledge of system behavior ψ.
  - Predicting future using ψ (Urgency constraint).
  - Urgency ensures that enforcer reacts ASAP.
- Properties φ and ψ are regular languages (modeled as automata).
- Enforcer synthesis.
- Algorithms implementing these mechanisms in constant on-line time.
- Proofs in Isabelle.
- Implementation: `https://github.com/SrinivasPinisetty/PredictiveRE`.

Future Work

- Predictive RE for real-time properties.
- Implementation, case-studies.

Conclusion and Future Work

Conclusion

- Introduced predictive RE framework.
  - A-priori knowledge of system behavior ψ.
  - Predicting future using ψ (Urgency constraint).
  - Urgency ensures that enforcer reacts ASAP.
- Properties φ and ψ are regular languages (modeled as automata).
- Enforcer synthesis.
- Algorithms implementing these mechanisms in constant on-line time.
- Proofs in Isabelle.
- Implementation: `https://github.com/SrinivasPinisetty/PredictiveRE`.

Future Work

- Predictive RE for real-time properties.
- Implementation, case-studies.

Thank you!!, Questions?