Runtime Enforcement of Reactive Systems using Synchronous Enforcers

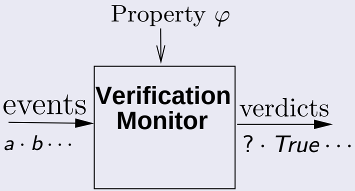
Srinivas Pinisetty[1,2], Partha Roop[3], Steven Smyth[5], Stavros Tripakis[1,4], Reinhard von Hanxleden[5]

Aalto University, Finland
University of Gothenburg, Sweden
University of Auckland, New Zealand
University of California, Berkeley
Kiel University, Germany

SPIN 2017, Santa Barbara, USA

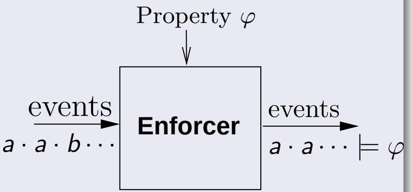
Runtime verification and enforcement

Runtime verification



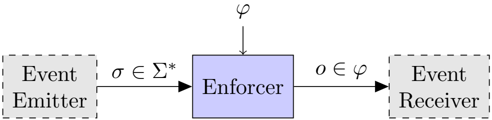
- Does σ satisfy φ ?
- Output: stream of **verdicts**.

Runtime enforcement



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

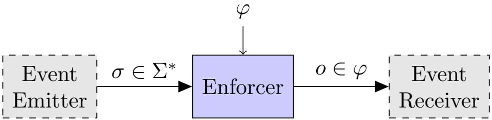
Runtime enforcement (previous work)



Enforcer for φ operating at runtime

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

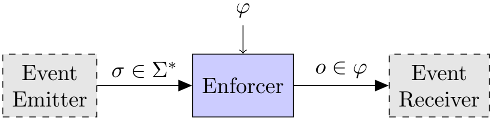
Runtime enforcement (previous work)



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 $o \models \varphi$.

Runtime enforcement (previous work)

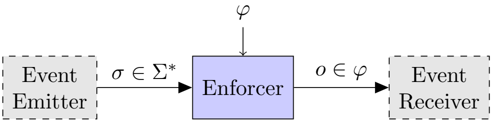


Enforcer for φ operating at runtime

- 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 $o \models \varphi$.

Power of the enforcement mechanism (what can an enforcer do)?

Runtime enforcement (previous work)



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 $o \models \varphi$.

Power of the enforcement mechanism (what can an enforcer do)?

Blocking/halting, delaying, and deleting events not suitable for reactive systems !

This work

- Focus on runtime enforcement for safety critical reactive systems.

This work

- Focus on runtime enforcement for safety critical reactive systems.
- Specifically reactive systems designed/developed using synchronous programming paradigm.

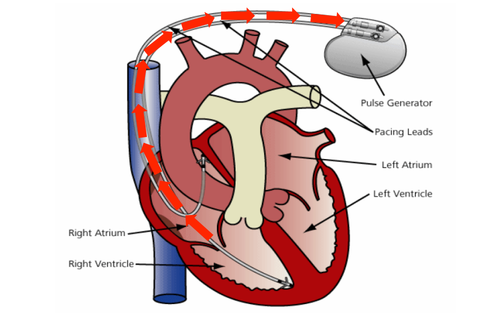
This work

- Focus on runtime enforcement for safety critical reactive systems.
- Specifically reactive systems designed/developed using synchronous programming paradigm.
- Bi-directional: monitor and enforce both the inputs and the outputs of a reactive system.

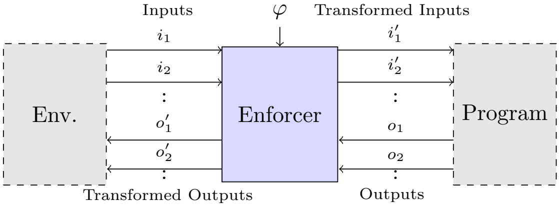
This work

- Focus on runtime enforcement for safety critical reactive systems.
- Specifically reactive systems designed/developed using synchronous programming paradigm.
- Bi-directional: monitor and enforce both the inputs and the outputs of a reactive system.

Applications: Medical devices, automotive, etc.



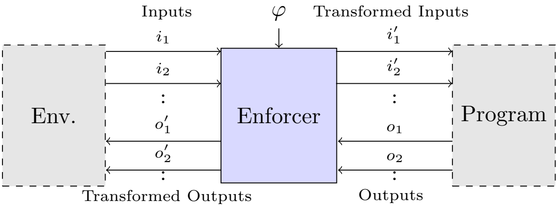
Bi-directional RE of synchronous systems problem (this work)



Enforcer for φ operating at runtime

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

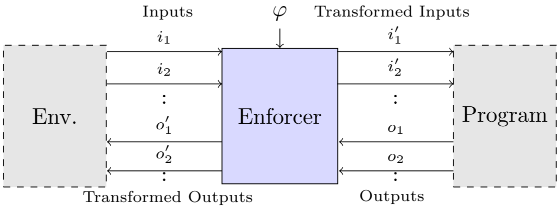
Bi-directional RE of synchronous systems problem (this work)



Enforcer for φ operating at runtime

- Given property φ (to enforce) defined as automaton.
- Automatically synthesize an enforcer.
  - Bi-directional enforcement (monitor and enforce both inputs and outputs).

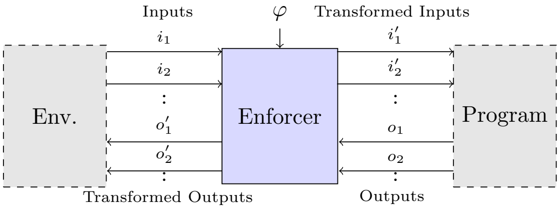
Bi-directional RE of synchronous systems problem (this work)



Enforcer for φ operating at runtime

- Given property φ (to enforce) defined as automaton.
- Automatically synthesize an enforcer.
  - Bi-directional enforcement (monitor and enforce both inputs and outputs).
  - Enforcer cannot halt the system or delay events, and must react instantaneously when an error is observed.

Bi-directional RE of synchronous systems problem (this work)



Enforcer for φ operating at runtime

- Given property φ (to enforce) defined as automaton.
- Automatically synthesize an enforcer.
  - Bi-directional enforcement (monitor and enforce both inputs and outputs).
  - Enforcer cannot halt the system or delay events, and must react instantaneously when an error is observed.

Enforcer should satisfy soundness, monotonicity, transparency, instantaneity, and causality constraints.

Outline

Outline

Some notations

Synchronous program

- $I = \{i_1, \cdots, i_n\}$, $O = \{o_1, \cdots, o_n\}$.
- Input alphabet $\Sigma_I = 2^I$, output alphabet $\Sigma_O = 2^O$ and $\Sigma = \Sigma_I \times \Sigma_O$.

Some notations

Synchronous program

- $I = \{i_1, \cdots, i_n\}$, $O = \{o_1, \cdots, o_n\}$.
- Input alphabet $\Sigma_I = 2^I$, output alphabet $\Sigma_O = 2^O$ and $\Sigma = \Sigma_I \times \Sigma_O$.
- A reaction is of the form (x_i, y_i), where $x_i \in \Sigma_I$ and $y_i \in \Sigma_O$.
- During each reaction, the program reacts to a set of inputs received from the environment to produce a set of outputs.

Some notations

Synchronous program

- $I = \{i_1, \cdots, i_n\}$, $O = \{o_1, \cdots, o_n\}$.
- Input alphabet $\Sigma_I = 2^I$, output alphabet $\Sigma_O = 2^O$ and $\Sigma = \Sigma_I \times \Sigma_O$.
- A reaction is of the form (x_i, y_i), where $x_i \in \Sigma_I$ and $y_i \in \Sigma_O$.
- During each reaction, the program reacts to a set of inputs received from the environment to produce a set of outputs.
- Execution of a program $\mathcal{P}$ is an infinite sequence of reactions.

Some notations

Synchronous program

- $I = \{i_1, \cdots, i_n\}$, $O = \{o_1, \cdots, o_n\}$.
- Input alphabet $\Sigma_I = 2^I$, output alphabet $\Sigma_O = 2^O$ and $\Sigma = \Sigma_I \times \Sigma_O$.
- A reaction is of the form (x_i, y_i), where $x_i \in \Sigma_I$ and $y_i \in \Sigma_O$.
- During each reaction, the program reacts to a set of inputs received from the environment to produce a set of outputs.
- Execution of a program $\mathcal{P}$ is an infinite sequence of reactions.

Example: $I = \{A, B\}$ and $O = \{R\}$

- $\Sigma_I = \{00, 01, 10, 11\}$
- $\Sigma_O = \{0, 1\}$
- $\Sigma = \{(00, 0), (00, 1), (01, 0), (01, 1), (10, 0), (10, 1), (11, 0), (11, 1)\}$

Properties

Safety properties

- $\Sigma = \Sigma_I \times \Sigma_O$, property $\varphi \subseteq \Sigma^*$.
- *Prefix-closed* properties (all prefixes of all words in $\mathcal{L}(\varphi)$ are also in $\mathcal{L}(\varphi)$).
- Property φ defined as a deterministic and complete safety automaton (SA) $\mathcal{A}_\varphi = (Q, q_0, q_v, \Sigma, \rightarrow)$.
- All the locations in Q except q_v (i.e., $Q \setminus \{q_v\}$) are accepting locations.

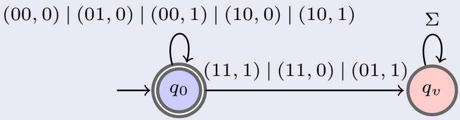
Properties

Safety properties

- $\Sigma = \Sigma_I \times \Sigma_O$, property $\varphi \subseteq \Sigma^*$.
- *Prefix-closed* properties (all prefixes of all words in $\mathcal{L}(\varphi)$ are also in $\mathcal{L}(\varphi)$).
- Property φ defined as a deterministic and complete safety automaton (SA) $\mathcal{A}_\varphi = (Q, q_0, q_v, \Sigma, \rightarrow)$.
- All the locations in Q except q_v (i.e., $Q \setminus \{q_v\}$) are accepting locations.

Example: Property defined as SA

$I = \{A, B\}$ and $O = \{R\}$



"A and B cannot happen simultaneously, and also B and R cannot happen simultaneously."

RE preliminaries (1)

- Consider $\sigma \in (x_1, y_1) \cdot (x_2, y_2) \cdots (x_n, y_n) \in \Sigma^*$.
  - σ_I: Projection on inputs $\sigma_I = x_1 \cdot x_2 \cdots x_n \in \Sigma_I$.
  - σ_O: Projection on outputs is $\sigma_O = y_1 \cdot y_2 \cdots y_n \in \Sigma_O$.

RE preliminaries (1)

- Consider $\sigma \in (x_1, y_1) \cdot (x_2, y_2) \cdots (x_n, y_n) \in \Sigma^*$.
  - σ_I: Projection on inputs $\sigma_I = x_1 \cdot x_2 \cdots x_n \in \Sigma_I$.
  - σ_O: Projection on outputs is $\sigma_O = y_1 \cdot y_2 \cdots y_n \in \Sigma_O$.
- $\mathcal{A}_{\varphi_I}$: From $\mathcal{A}_\varphi$, $\mathcal{A}_{\varphi_I}$ is obtained by ignoring outputs on the transitions.

RE preliminaries (2)

Input SA (obtained by projecting on inputs)

Given property $\varphi \subseteq \Sigma^*$, defined as SA $\mathcal{A}_\varphi = (Q, q_0, q_v, \Sigma, \rightarrow)$;

- $\mathcal{A}_{\varphi_I} = (Q, q_0, q_v, \Sigma_I, \rightarrow_I)$ is obtained from $\mathcal{A}_\varphi$ by ignoring outputs on the transitions.

RE preliminaries (2)

Input SA (obtained by projecting on inputs)

Given property $\varphi \subseteq \Sigma^*$, defined as SA $\mathcal{A}_\varphi = (Q, q_0, q_v, \Sigma, \rightarrow)$;

- $\mathcal{A}_{\varphi_I} = (Q, q_0, q_v, \Sigma_I, \rightarrow_I)$ is obtained from $\mathcal{A}_\varphi$ by ignoring outputs on the transitions.
- Every transition $q \xrightarrow{(x,y)} q'$ in $\mathcal{A}_\varphi$ is replaced with transition $q \xrightarrow{x}_I q'$ in $\mathcal{A}_{\varphi_I}$.

RE preliminaries (2)

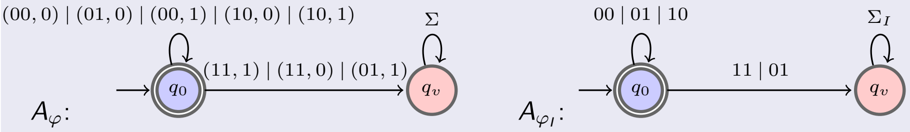
Input SA (obtained by projecting on inputs)

Given property $\varphi \subseteq \Sigma^*$, defined as SA $\mathcal{A}_\varphi = (Q, q_0, q_v, \Sigma, \rightarrow)$;

- $\mathcal{A}_{\varphi_I} = (Q, q_0, q_v, \Sigma_I, \rightarrow_I)$ is obtained from $\mathcal{A}_\varphi$ by ignoring outputs on the transitions.
- Every transition $q \xrightarrow{(x,y)} q'$ in $\mathcal{A}_\varphi$ is replaced with transition $q \xrightarrow{x}_I q'$ in $\mathcal{A}_{\varphi_I}$.

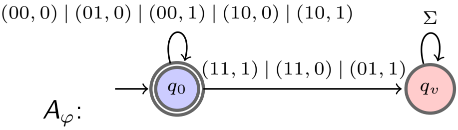
Example: SA (left) and its input SA (right)

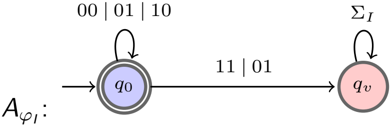
- $I = \{A, B\}$, and $O = \{R\}$,



"A and B cannot happen simultaneously and also B and R cannot happen simultaneously".

RE preliminaries (Edit functions)

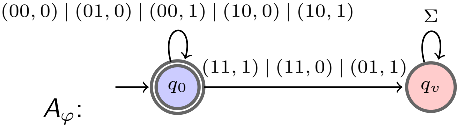


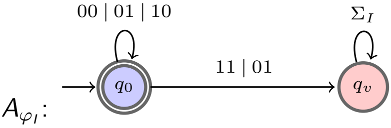


$$\mathsf{editI}_{\mathcal{A}_{\varphi_I}}(q) = \{x \in \Sigma_I : q \xrightarrow{x}_I q' \wedge q' \neq q_v\}.$$

Example: $\mathsf{editI}_{\mathcal{A}_{\varphi_I}}(q_0) = \{00, 01, 10\}$.

RE preliminaries (Edit functions)





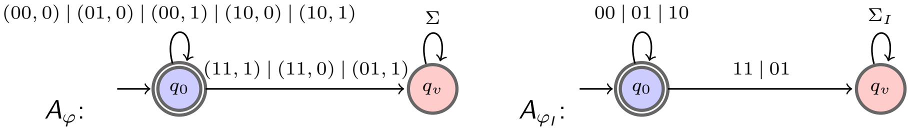
$$\text{editI}_{\mathcal{A}_{\varphi_I}}(q) = \{x \in \Sigma_I : q \xrightarrow{x}_I q' \wedge q' \neq q_v\}.$$

Example: $\text{editI}_{\mathcal{A}_{\varphi_I}}(q_0) = \{00, 01, 10\}$.

$$\text{editO}_{\mathcal{A}_\varphi}(q, x) = \{y \in \Sigma_O : q \xrightarrow{(x,y)} q' \wedge q' \neq q_v\}.$$

Example: $\text{editO}_{\mathcal{A}_\varphi}(q_0, 01) = \{0\}$.

RE preliminaries (Edit functions)



$\text{editI}_{\mathcal{A}_{\varphi_I}}(q) = \{x \in \Sigma_I : q \xrightarrow{x}_I q' \wedge q' \neq q_v\}$.

Example: $\text{editI}_{\mathcal{A}_{\varphi_I}}(q_0) = \{00, 01, 10\}$.

$\text{editO}_{\mathcal{A}_\varphi}(q, x) = \{y \in \Sigma_O : q \xrightarrow{(x,y)} q' \wedge q' \neq q_v\}$.

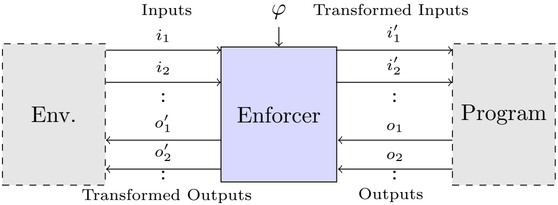
Example: $\text{editO}_{\mathcal{A}_\varphi}(q_0, 01) = \{0\}$.

- nondet–$\text{editI}_{\mathcal{A}_{\varphi_I}}(q)$: An element chosen randomly from $\text{editI}_{\mathcal{A}_{\varphi_I}}(q)$.
- nondet–$\text{editO}_{\mathcal{A}_\varphi}(\sigma, x)$: An element chosen randomly from $\text{editO}_{\mathcal{A}_\varphi}(q, x)$.

Outline

Enforcer for φ

Given property φ (defined as SA):



What can an enforcer do?

- CAN edit an input-output event when necessary.
- CANNOT delay, block, suppress events.

Formal problem definition

Preliminaries (recall)

- I: set of inputs, O: set of outputs.
- $\Sigma_I = 2^I$, $\Sigma_O = 2^O$, and $\Sigma = \Sigma_I \times \Sigma_O$.
- Event (reaction): (x_i, y_i) where $x_i \in \Sigma_i$ and $y_i \in \Sigma_O$.
- Word σ: $(x_0, y_0) \cdot (x_1, y_1) \cdots \in \Sigma^*$.
- Property φ: $\varphi \subseteq \Sigma^*$.

Given φ, synthesize an enforcer $E_\varphi : \Sigma^* \rightarrow \Sigma^*$ that satisfies:

- **Soundness**
- **Monotonicity**
- **Transparency**
- **Instantaneity**
- **Causality**

Soundness

Output is correct (satisfies φ)

$$\forall \sigma \in \Sigma^* : E_\varphi(\sigma) \models \varphi.$$

Monotonicity

Modify output only by appending new events

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

Transparency

Change only when necessary

$$\forall \sigma \in \Sigma^*, \forall x \in \Sigma_I, \forall y \in \Sigma_O:$$
$$E_\varphi(\sigma) \cdot (x, y) \models \varphi \implies E_\varphi(\sigma \cdot (x, y)) = E_\varphi(\sigma) \cdot (x, y).$$

Transparency

Change only when necessary

$$\forall \sigma \in \Sigma^*, \forall x \in \Sigma_I, \forall y \in \Sigma_O:$$
$$E_\varphi(\sigma) \cdot (x, y) \models \varphi \implies E_\varphi(\sigma \cdot (x, y)) = E_\varphi(\sigma) \cdot (x, y).$$

Lemma (Transparency is stronger)

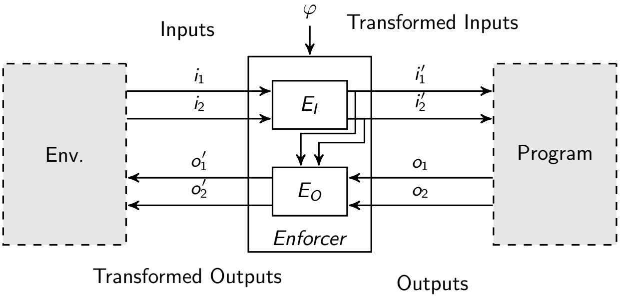
$$\textit{Transparency} \implies (\forall \sigma \in \Sigma^* : \sigma \models \varphi \Rightarrow E_\varphi(\sigma) = \sigma).$$

Instantaneity

Length preserving (cannot delay, insert and suppress events)

$$\forall \sigma \in \Sigma^* : |\sigma| = |E_\varphi(\sigma)|$$

Causality



Input from the env. $\rightarrow$ Output from the program

$$\forall \sigma \in \Sigma^*, \forall x \in \Sigma_I, \forall y \in \Sigma_O,$$
$$\exists x' \in \mathrm{editI}_{\varphi_I}(E_\varphi(\sigma)_I), \exists y' \in \mathrm{editO}_\varphi(E_\varphi(\sigma), x') :$$
$$E_\varphi(\sigma \cdot (x, y)) = E_\varphi(\sigma) \cdot (x', y').$$

Outline

Enforceable safety properties

Enforceability

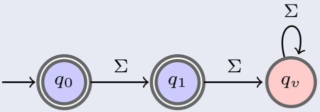
Let $\varphi \subseteq \Sigma^*$ be a property. We say that φ is *enforceable* iff an enforcer E_φ for φ exists.

Enforceable safety properties

Enforceability

Let $\varphi \subseteq \Sigma^*$ be a property. We say that φ is *enforceable* iff an enforcer E_φ for φ exists.

A non-enforceable safety property



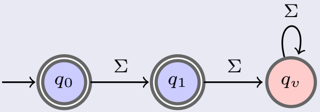
- $I = \{A\}, O = \{B\}$.
- Consider any input word of length greater than 1 (e.g., $(0,1) \cdot (1,1)$).

Enforceable safety properties

Enforceability

Let $\varphi \subseteq \Sigma^*$ be a property. We say that φ is *enforceable* iff an enforcer E_φ for φ exists.

A non-enforceable safety property



- $I = \{A\}, O = \{B\}$.
- Consider any input word of length greater than 1 (e.g., $(0,1) \cdot (1,1)$).

Condition for enforceability

A property φ defined as SA $A_\varphi = (Q, q_0, q_v, \Sigma, \rightarrow)$ is enforceable iff

$$\forall q \in Q, q \neq q_v \implies \exists (x,y) \in \Sigma : q \xrightarrow{(x,y)} q' \wedge q' \neq q_v$$

Transformation of non-enforceable properties

Transformation algorithm

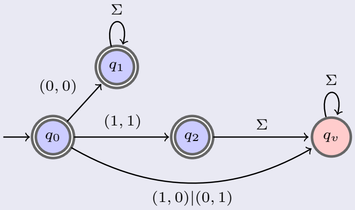
$sub(\mathcal{A}_\varphi) \leftarrow \mathcal{A}_\varphi = (Q, q_0, q_v, \Sigma, \rightarrow)$
while $\exists q \in Q \setminus \{q_v\} : \forall (x, y) \in \Sigma, q \xrightarrow{(x,y)} q_v$ **do**
  for all $q \in Q \setminus \{q_v\}$ **do**
    if $\forall (x, y) \in \Sigma, q \xrightarrow{(x,y)} q_v$ **then**
      if $q = q_0$ **then**
        RETURN(NONE)
      else
        remove(q)
      end if
    end if
  end for
end while
RETURN(sub($\mathcal{A}_\varphi$))

- Transformation algorithm is complete.
- Some behaviors excluded (i.e., $\mathcal{L}(sub(\mathcal{A}_\varphi)) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$).
- Removal of behaviors done minimally.

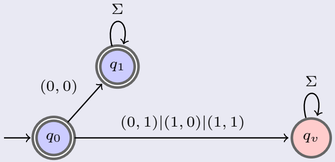
Transformation of non-enforceable properties (Examples)

A non-enforceable property that **can** be transformed into an enforceable property

A_φ:



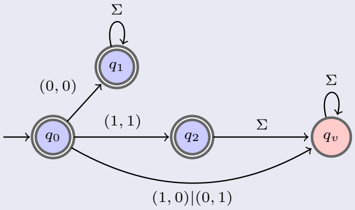
$Sub(A_\varphi)$:



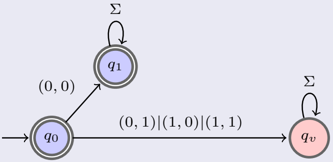
Transformation of non-enforceable properties (Examples)

A non-enforceable property that **can** be transformed into an enforceable property

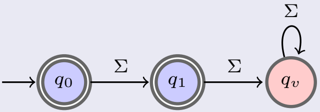
A_φ:



$Sub(A_\varphi)$:



A non-enforceable property that **cannot** be transformed into an enforceable property



Outline

Enforcement algorithm (1)

Input: $A_\varphi = (Q, q_0, q_v, \Sigma, \rightarrow)$.
$A_{\varphi_I} = (Q_I, q_{0_I}, q_{v_I}, \Sigma_I, \rightarrow_I)$ (Obtained from A_φ by ignoring outputs.)

Online algorithm

initialize tick/time, automata current states;
while *True* **do**
    READ-input-channels;
    EDIT-input-if-necessary;
    READ-output-channels (after invoking program);
    EDIT-output-if-necessary;
    UPDATE-automata-current-states;
end

Enforcement algorithm (2)

Enforcer

$t \leftarrow 0$
$q \leftarrow q_0$
while true **do**
    $x_t \leftarrow \text{read\_in\_chan}()$
    if $\exists q' \in Q : q \xrightarrow{x_t}_I q' \wedge q' \neq q_v$ **then**
        $x'_t \leftarrow x_t$
    else
        $x'_t \leftarrow \text{nondet–editI}_{\mathcal{A}_{\varphi_I}}(q)$
    end if
    $\text{ptick}(x'_t)$
    $y_t \leftarrow \text{read\_out\_chan}()$
    if $\exists q' \in Q : q \xrightarrow{(x'_t, y_t)} q' \wedge q' \neq q_v$ **then**
        $y'_t \leftarrow y_t$
    else
        $y'_t \leftarrow \text{nondet–editO}_{\mathcal{A}_{\varphi}}(q, x'_t)$
    end if
    $\text{release}((x'_t, y'_t))$
    $q \leftarrow q'$ where $q \xrightarrow{(x'_t, y'_t)} q' \wedge q' \neq q_v$
    $t \leftarrow t + 1$
end while

Enforcement algorithm (3)

Definition (E^*_φ)

Let $\sigma = (x_1, y_1) \cdots (x_k, y_k) \in \Sigma^*$ be a word received by the algorithm, and let $E^*_\varphi(\sigma) = (x'_1, y'_1) \cdots (x'_k, y'_k)$, where (x'_t, y'_t) is the pair of events output by the enforcement algorithm in Step 17, for $t = 1, ..., k$.

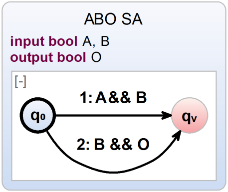
Theorem (Correctness of the enforcement algorithm)

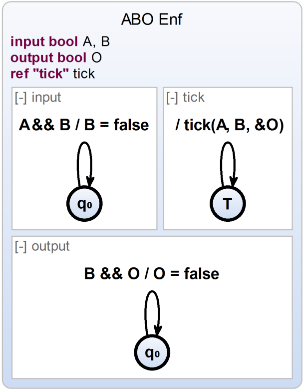
Given any safety property φ defined as SA $\mathcal{A}_\varphi$ that satisfies enforceability condition, the function E^*_φ defined above is an enforcer for φ, that is, it satisfies soundness, transparency, monotonicity, instantainety, and causality constraints.

Outline

Application to the SCCharts synchronous language

Example: Property and its enforcer





Results

Examples	Tick (LoC)	#: Properties	Enf. (LoC)	Time (ms)	Time w/ Enf. (ms)	Incr. (%)
ABRO	23	1	21	0.001208	0.001565	29.55
ABO	28	1	21	0.000998	0.001368	37.10
Reactor	32	2	32	0.001587	0.002137	34.61
Faulty Heart Model	43	2	40	0.001346	0.001869	38.85
Simple Heart Model	76	2	40	0.002175	0.002825	29.86
Traffic Light	171	3	41	0.004039	0.004707	16.53
Pacemaker	271	2	35	0.007302	0.008318	13.91
FHM + Pacemaker	314	2	35	0.009195	0.010306	12.08

Outline

Conclusion and Future Work

Conclusion

- Introduced bi-directional RE framework for reactive systems (modelled using the synchronous approach).
  - Formalize bi-directional RE problem.
  - Enforceability conditions (characterize set of safety properties which can be enforced).
  - Enforcer synthesis algorithm.
- Implemented the proposed enforcer synthesis algorithm for the SCCharts synchronous language.
- Applicability of the proposed approach by enforcing policies over a synchronous pacemaker model.

Conclusion and Future Work

Conclusion

- Introduced bi-directional RE framework for reactive systems (modelled using the synchronous approach).
  - Formalize bi-directional RE problem.
  - Enforceability conditions (characterize set of safety properties which can be enforced).
  - Enforcer synthesis algorithm.
- Implemented the proposed enforcer synthesis algorithm for the SCCharts synchronous language.
- Applicability of the proposed approach by enforcing policies over a synchronous pacemaker model.

Future Work

- Practical setting of implantable pacemakers.
- Enforce other regular properties, discrete-time, etc.

Conclusion and Future Work

Conclusion

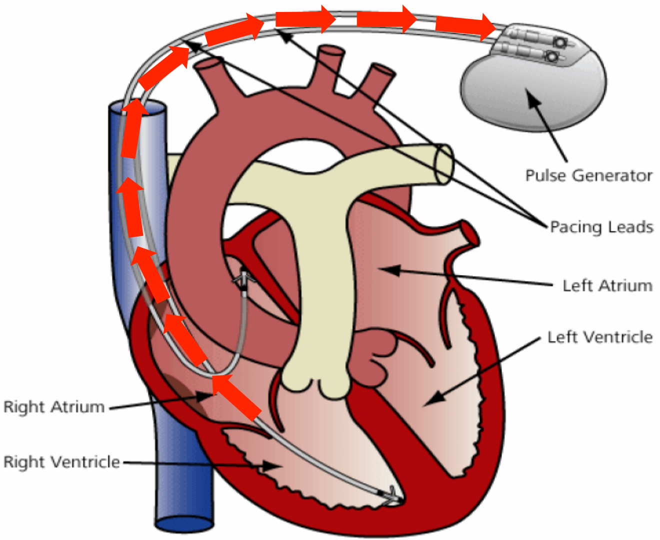
- Introduced bi-directional RE framework for reactive systems (modelled using the synchronous approach).
  - Formalize bi-directional RE problem.
  - Enforceability conditions (characterize set of safety properties which can be enforced).
  - Enforcer synthesis algorithm.
- Implemented the proposed enforcer synthesis algorithm for the SCCharts synchronous language.
- Applicability of the proposed approach by enforcing policies over a synchronous pacemaker model.

Future Work

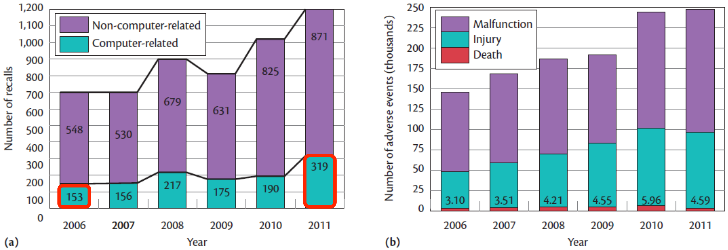
- Practical setting of implantable pacemakers.
- Enforce other regular properties, discrete-time, etc.

Thank you!!, Questions?

Implantable pacemakers



Adverse events



	Class I: high risk	Class II: medium risk	Class III: low risk	Total recalls	Number of devices
Software	14 (33.3%)	718 (65.6%)	46 (75.3%)	778 (64.3%)	2,303,441 (19.2%)
Hardware	8 (19.0%)	158 (14.4%)	13 (27.4%)	179 (14.8%)	4,228,133 (35.2%)
Other	10 (23.8%)	124 (11.3 %)	8 (12.3%)	142 (11.7%)	2,831,048 (23.5%)
Battery	8 (19.0%)	57 (5.2%)	5 (6.8%)	70 (5.8%)	2,385,613(19.8%)
I/O	2 (4.8%)	38 (3.5%)	1 (2.7%)	41 (3.4%)	276,601 (2.3%)
Total recalls	42 (3.5%)	1,095 (90.5%)	73 (6.0%)	1,210	12,024,836

[Ref.]: Alemzadeh, H., Iyer, R.K., Kalbarczyk, Z., Raman, J., "Analysis of Safety-Critical Computer Failures in Medical Devices", *Security and Privacy* , IEEE , vol.11, no.4. pp.14,26. July-Aug, 2013.

Approaches to enhance pacemaker software

- Two key CS related initiatives: http://cybercardia.cs.stonybrook.edu, and Marta Kwiatkowska's group in Oxford.
- Model-based approach: Modeling and verification of a dual chamber implantable pacemaker, *Jiang, Pajic, Moarref, Alur, Mangaram.* TACAS 2012
- Testing: Heart-on-a-chip: A closed-loop testing platform for implantable pacemakers *Jiang, Radhakrishnan, Sampath, Sarode, Mangharam.* CyPhy 2013
- Requirements-Centric Closed-Loop Validation of Implantable Cardiac Devices. *Weiwei Ai, Nitish Patel and Partha Roop.* DATE '16.
- Except the work of Ai et al., others consider a static model of the heart during closed-loop testing / model checking.
- Focus of the current work is on *run-time enforcement*, where a dynamically evolving heart model and a pacemaker can be used for run-time verification and enforcement.

When input σ satisfies φ

Transparency': $\forall \sigma \in \Sigma^* : \sigma \in \varphi \Rightarrow E_\varphi(\sigma) = \sigma$

Transparency' means that when the input sequence σ satisfies φ, then σ will be the output of the enforcer.

Lemma (*Transparency* $\implies$ *Transparency'*)

$(\forall \sigma \in \Sigma^*, \forall x \in \Sigma_I, \forall y \in \Sigma_O : E_\varphi(\sigma) \cdot (x, y) \models \varphi \implies E_\varphi(\sigma \cdot (x, y)) = E_\varphi(\sigma) \cdot (x, y))$
$\implies$
$(\forall \sigma \in \Sigma^* : \sigma \in \varphi \Rightarrow E_\varphi(\sigma) = \sigma)$.

Example (Transparency is stronger)

- $I = \{A, B\}$, $O = \{O\}$, Property φ: A and B cannot happen simultaneously.

σ	$E_\varphi(\sigma)$	TR	TR'
$(01, -)$	$(01, -)$	✓	✓
$(01, -) \cdot (11, -)$	$(01, -) \cdot (10, -)$	✓	✓
$(01, -) \cdot (11, -) \cdot (01, -)$	$(01, -) \cdot (10, -) \cdot (10, -)$	✗	✓
$(01, -) \cdot (11, -) \cdot (01, -)$	$(01, -) \cdot (10, -) \cdot (01, -)$	✓	✓

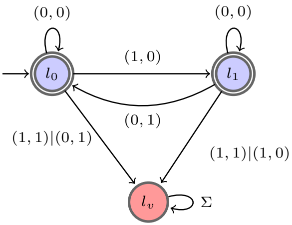
Results

Examples	Tick (LoC)	φ: in-out	Enf. (LoC)	Time (ms)	Time w/ Enf. (ms)	Incr. (%)
Null	0	0-0	0	0.000654	0.000752	14.98
ABRO	23	1-0	21	0.001208	0.001565	29.55
ABO	28	1-0	21	0.000998	0.001368	37.10
Reactor	32	1-1	32	0.001587	0.002137	34.61
Faulty Heart Model	43	1-1	40	0.001346	0.001869	38.85
Simple Heart Model	76	1-1	40	0.002175	0.002825	29.86
Traffic Light	171	0-3	41	0.004039	0.004707	16.53
Pacemaker	271	1-1	35	0.007302	0.008318	13.91
FHM + Pacemaker	314	1-1	35	0.009195	0.010306	12.08

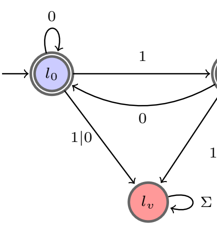
Example

$I = \{A\},\ O = \{R\}$

A_φ:



A_{φ_I}:



t	x	x'	y	y'	Loc	EnfAct
0	ϵ	ϵ	ϵ	ϵ	l_0	-
1	0	0	1	0	l_0	fwd$_I$, edt$_O$
2	1	1	0	0	l_1	fwd$_I$, fwd$_O$
3	1	0	0	0	l_1	edt$_I$, fwd$_O$
4	0	0	1	1	l_0	fwd$_I$, fwd$_O$