FORTE 2017

Monday 19th June 2017
Neuchâtel, Suisse

Learning-Based Compositional Parameter Synthesis for Event-Recording Automata

Étienne André[1] and Shang-Wei Lin[2]

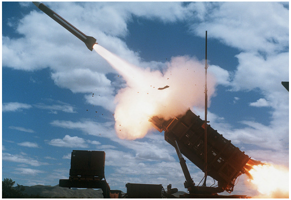
[1]LIPN, Université Paris 13, CNRS, France
[2]SCSE, Nanyang Technological University, Singapore

PROJET FINANCÉ PAR L'ANR - PROJECT FUNDED BY THE ANR

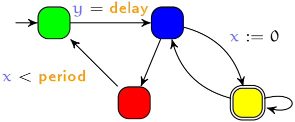
Context: Critical distributed systems

- Need for early bug detection
  - Bugs discovered when final testing: expensive

$\leadsto$ Need for a thorough modeling and verification phase



Timed model checking (1/2)

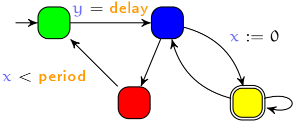


A model of the system

 is unreachable

A property to be satisfied

Timed model checking (1/2)



$\overset{?}{\models}$

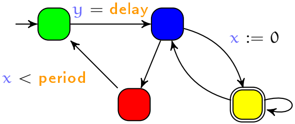
 is unreachable

A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Timed model checking (1/2)



$\overset{?}{\models}$

 is unreachable

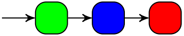
A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Yes

No



Counterexample

Timed model checking (2/2)

- Timed systems are characterized by a set of timing constants
  - "The packet transmission lasts for 50 ms"
  - "The sensor reads the value every 10 s"
- Powerful model checking tools, e.g.:
  - UPPAAL [Larsen et al., 1997]
  - PAT [Sun et al., 2009]

Beyond timed model checking: parameter synthesis

- Verification for one set of constants does not usually guarantee the correctness for other values

- Challenges
  - Numerous verifications: is the system correct for any value within $[40; 60]$?
  - Optimization: until what value can we increase 10?
  - Robustness [Markey, 2011]: What happens if 50 is implemented with 49.99?
  - System incompletely specified: Can I verify my system even if I don't know the period value with full certainty?

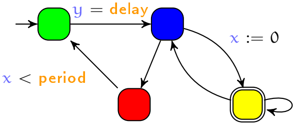
Beyond timed model checking: parameter synthesis

- Verification for one set of constants does not usually guarantee the correctness for other values

- Challenges
  - Numerous verifications: is the system correct for any value within $[40; 60]$?
  - Optimization: until what value can we increase 10?
  - Robustness [Markey, 2011]: What happens if 50 is implemented with 49.99?
  - System incompletely specified: Can I verify my system even if I don't know the period value with full certainty?

- Parameter synthesis
  - Consider that timing constants are unknown constants (parameters)

timed model checking



$\overset{?}{\models}$

 is unreachable

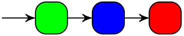
A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

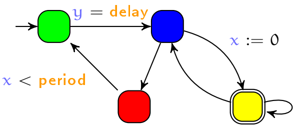
Yes

No



Counterexample

Parametric timed model checking



$\overset{?}{\models}$

 is unreachable

A model of the system

A property to be satisfied

- Question: for what values of the parameters does the model of the system satisfy the property?

Yes if...

$$2\mathsf{delay} > \mathsf{period} \\ \wedge\ \mathsf{period} < 20.46$$

Outline

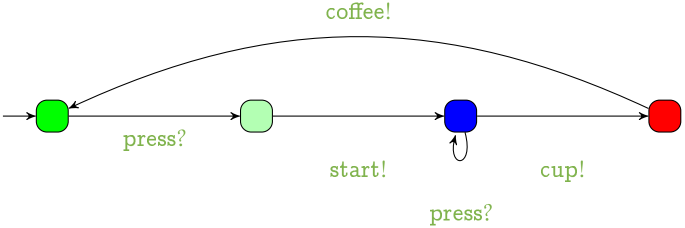
Outline

Event-recording automata (ERA)

- Finite state automaton (sets of locations)

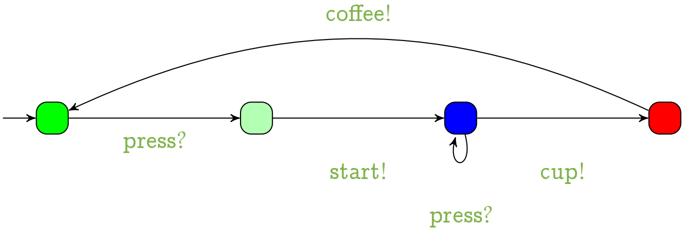
Event-recording automata (ERA)

- Finite state automaton (sets of locations and actions)



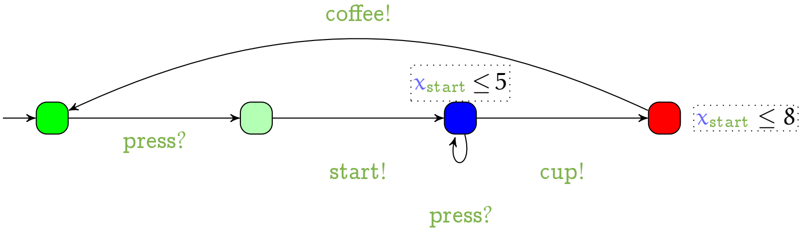
Event-recording automata (ERA)

- Finite state automaton (sets of locations and actions) augmented with a set X of clocks, one clock for each action [Alur et al., 1999]
  - Clocks: Real-valued variables evolving linearly at the same rate
  - A (strict) subclass of timed automata [Alur and Dill, 1994]



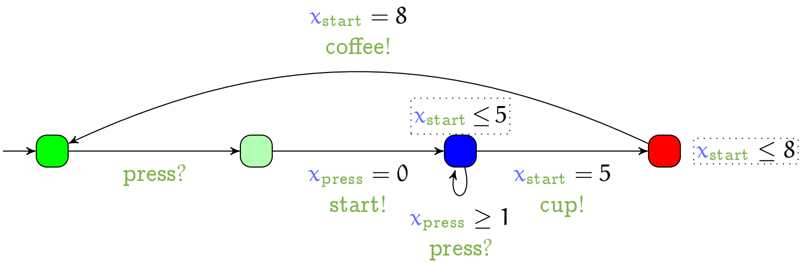
Event-recording automata (ERA)

- Finite state automaton (sets of locations and actions) augmented with a set X of clocks, one clock for each action [Alur et al., 1999]
  - Clocks: Real-valued variables evolving linearly at the same rate
  - A (strict) subclass of timed automata [Alur and Dill, 1994]
- Features
  - Location invariant: constraint to be verified to stay at a location



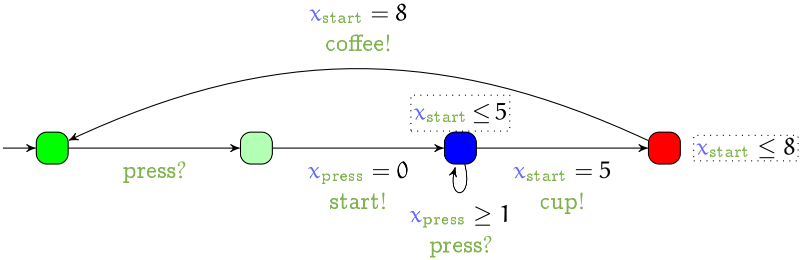
Event-recording automata (ERA)

- Finite state automaton (sets of locations and actions) augmented with a set X of clocks, one clock for each action [Alur et al., 1999]
  - Clocks: Real-valued variables evolving linearly at the same rate
  - A (strict) subclass of timed automata [Alur and Dill, 1994]
- Features
  - Location invariant: constraint to be verified to stay at a location
  - Transition guard: constraint to be verified to enable a transition



Event-recording automata (ERA)

- Finite state automaton (sets of locations and actions) augmented with a set X of clocks, one clock for each action [Alur et al., 1999]
  - Clocks: Real-valued variables evolving linearly at the same rate
  - A (strict) subclass of timed automata [Alur and Dill, 1994]
- Features
  - Location invariant: constraint to be verified to stay at a location
  - Transition guard: constraint to be verified to enable a transition
  - For each action a, the corresponding clock x_a is (implicitly) reset

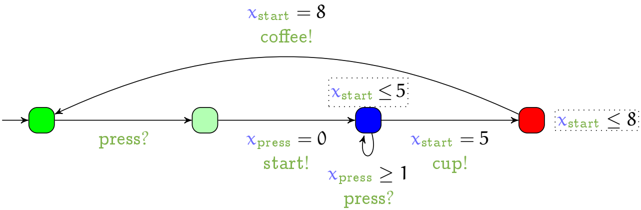


Concrete semantics of event-recording automata

- Concrete state of an ERA: pair (l, w), where
  - l is a location,
  - w is a valuation of each clock

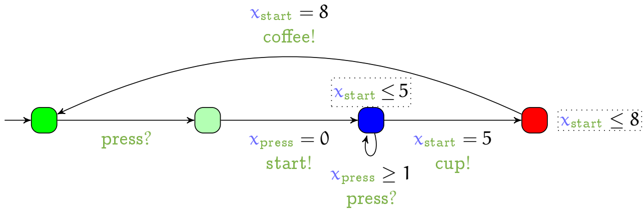
- Concrete run: alternating sequence of concrete states and actions or time elapse

Examples of concrete runs



- Possible concrete runs for the coffee machine

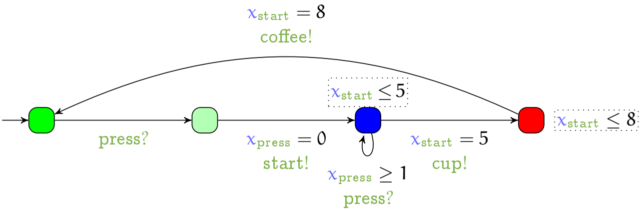
Examples of concrete runs



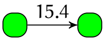
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

x_{press} 0
x_{start} 0

Examples of concrete runs

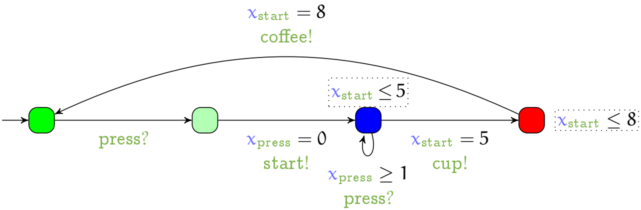


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

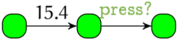


x_{press}	0	15.4
x_{start}	0	15.4

Examples of concrete runs

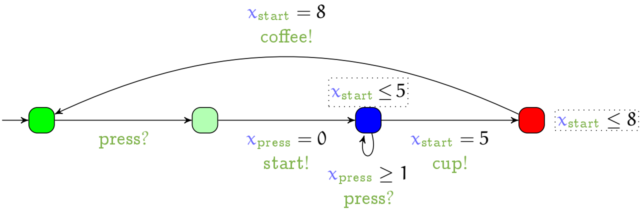


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



x_{press}	0	15.4	0
x_{start}	0	15.4	15.4

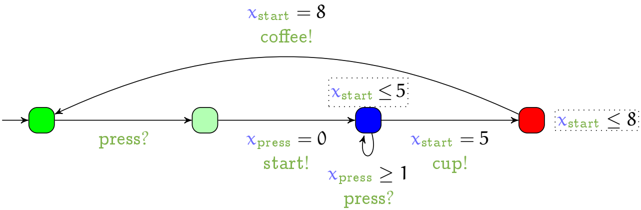
Examples of concrete runs



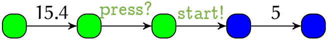
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

x_{press}	0	15.4	0	0
x_{start}	0	15.4	15.4	0

Examples of concrete runs

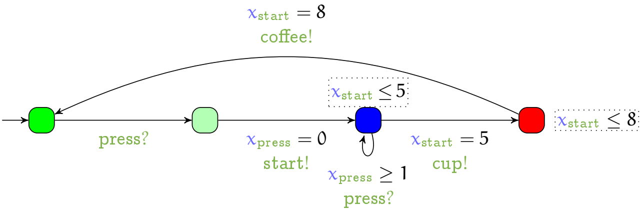


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



x_{press}	0	15.4	0	0	5
x_{start}	0	15.4	15.4	0	5

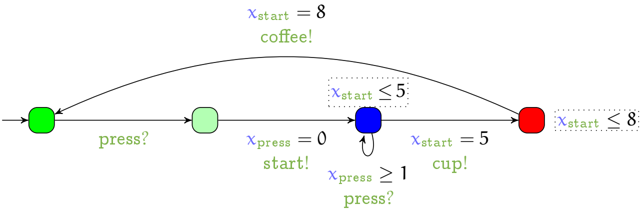
Examples of concrete runs



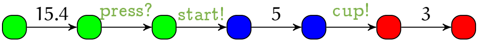
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

x_{press}	0	15.4	0	0	5	5
x_{start}	0	15.4	15.4	0	5	5

Examples of concrete runs

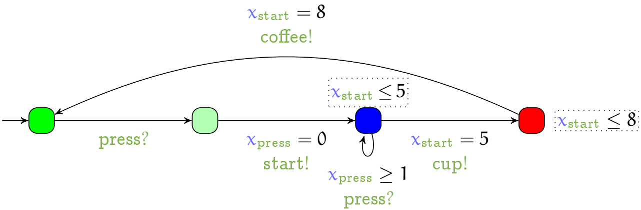


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

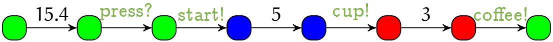


x_{press}	0	15.4	0	0	5	5	8
x_{start}	0	15.4	15.4	0	5	5	8

Examples of concrete runs

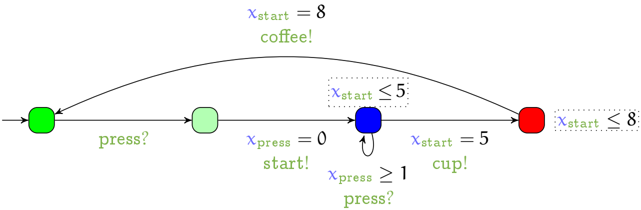


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

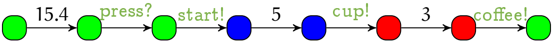


x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

Examples of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar

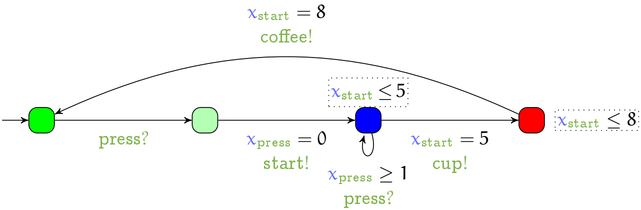


x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

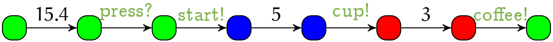
  - Coffee with 2 doses of sugar

x_{press}	0
x_{start}	0

Examples of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar

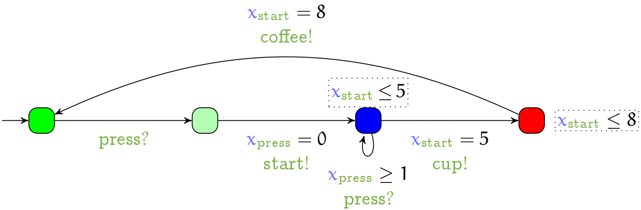


x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

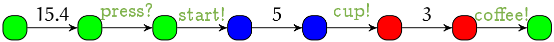
  - Coffee with 2 doses of sugar

x_{press}	0	0
x_{start}	0	0

Examples of concrete runs

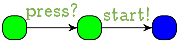


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



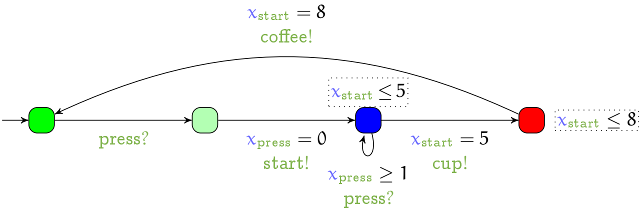
x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

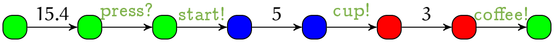


x_{press}	0	0	0
x_{start}	0	0	0

Examples of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar

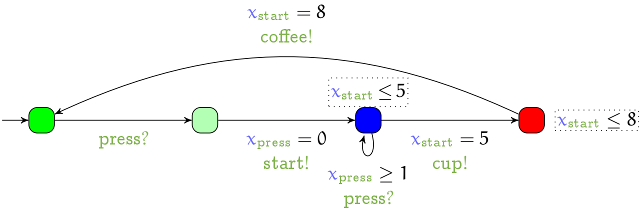


x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

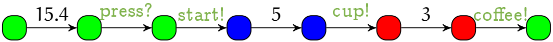
  - Coffee with 2 doses of sugar

x_{press}	0	0	0	1.5
x_{start}	0	0	0	1.5

Examples of concrete runs

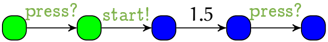


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



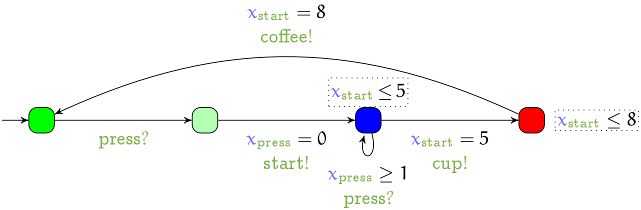
x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

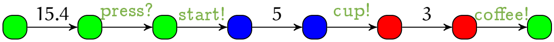


x_{press}	0	0	0	1.5	0
x_{start}	0	0	0	1.5	1.5

Examples of concrete runs

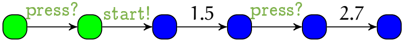


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



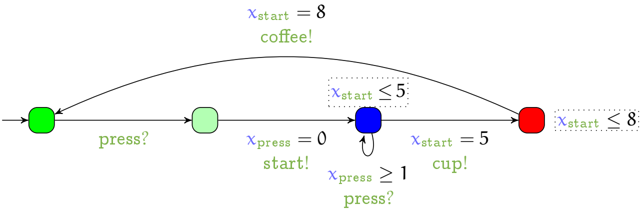
x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

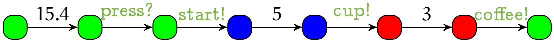


x_{press}	0	0	0	1.5	0	2.7
x_{start}	0	0	0	1.5	1.5	4.2

Examples of concrete runs

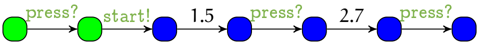


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



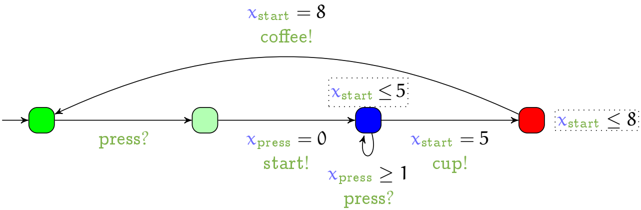
x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

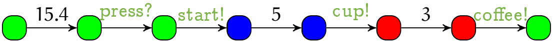


x_{press}	0	0	0	1.5	0	2.7	0
x_{start}	0	0	0	1.5	1.5	4.2	4.2

Examples of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar

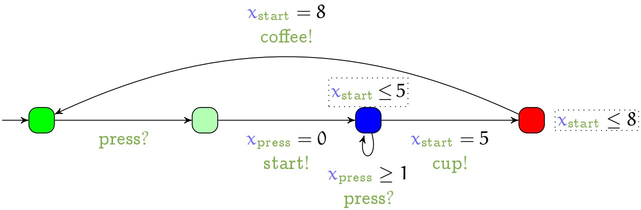


x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

x_{press}	0	0	0	1.5	0	2.7	0	0.8
x_{start}	0	0	0	1.5	1.5	4.2	4.2	5

Examples of concrete runs



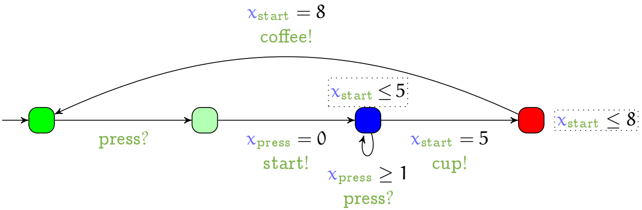
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

		→ 15.4 →		→ press? →		→ start! →		→ 5 →		→ cup! →		→ 3 →		→ coffee! →	
x_{press}	0		15.4		0		0		5		5		8		8
x_{start}	0		15.4		15.4		0		5		5		8		8

  - Coffee with 2 doses of sugar

		→ press? →		→ start! →		→ 1.5 →		→ press? →		→ 2.7 →		→ press? →		→ 0.8 →		→ cup! →	
x_{press}	0		0		0		1.5		0		2.7		0		0.8		0.8
x_{start}	0		0		0		1.5		1.5		4.2		4.2		5		5

Examples of concrete runs



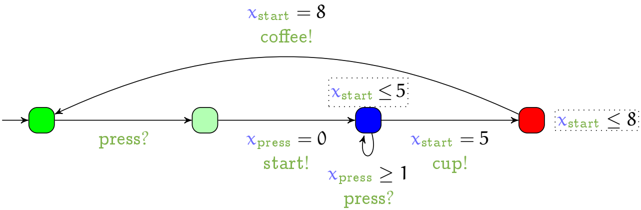
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

		15.4		press?		start!		5		cup!		3		coffee!	
x_{press}	0		15.4		0		0		5		5		8		8
x_{start}	0		15.4		15.4		0		5		5		8		8

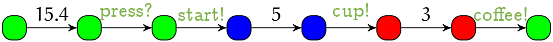
  - Coffee with 2 doses of sugar

		press?		start!		1.5		press?		2.7		press?		0.8		cup!		3	
x_{press}	0		0		0		1.5		0		2.7		0		0.8		0.8		3.8
x_{start}	0		0		0		1.5		1.5		4.2		4.2		5		5		8

Examples of concrete runs

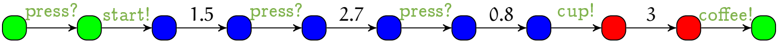


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



x_{press}	0	15.4	0	0	5	5	8	8
x_{start}	0	15.4	15.4	0	5	5	8	8

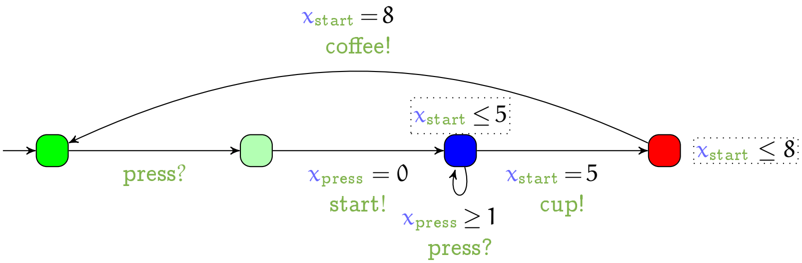
  - Coffee with 2 doses of sugar



x_{press}	0	0	0	1.5	0	2.7	0	0.8	0.8	3.8	3.8
x_{start}	0	0	0	1.5	1.5	4.2	4.2	5	5	8	8

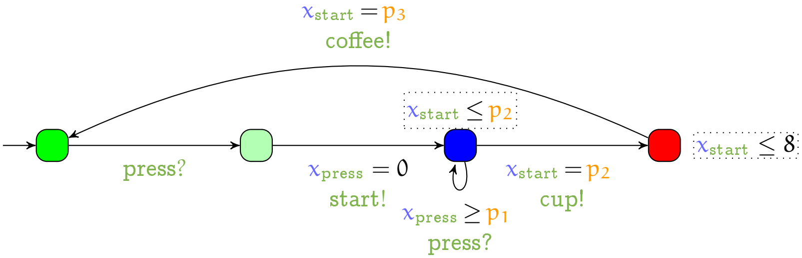
Parametric event-recording automata (PERA)

- Event-recording automata (sets of locations, actions and clocks)



Parametric event-recording automata (PERA)

- Event-recording automata (sets of locations, actions and clocks) augmented with a set P of parameters
  - Parameters: Unknown constants used in guards and invariants
  - Extension in the spirit of parametric timed automata [Alur et al., 1993]

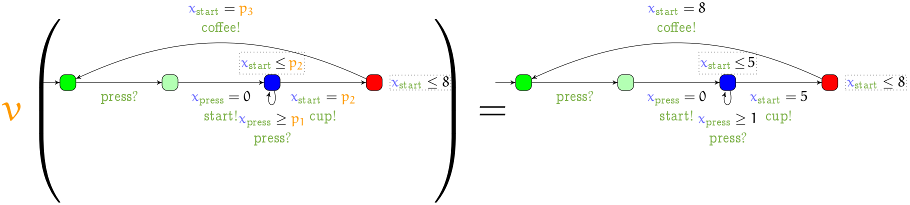


Valuation of a PERA

- Given a PERA A and a parameter valuation v, we denote by $v(A)$ the (non-parametric) event-recording automaton where all parameters are valuated by v

Valuation of a PERA

- Given a PERA A and a parameter valuation v, we denote by $v(A)$ the (non-parametric) event-recording automaton where all parameters are valuated by v



$$\text{with } v : \begin{cases} p_1 & \rightarrow & 1 \\ p_2 & \rightarrow & 5 \\ p_3 & \rightarrow & 8 \end{cases}$$

Problems

Reachability-emptiness

Input: a PERA A, a location l
Question: is the set of valuations v for which a runs reaches l in $v(A)$ empty?

Reachability-synthesis

Input: a PERA A, a location l
Question: synthesize all valuations v for which a runs reaches l in $v(A)$

An undecidability result

Theorem (Undecidability)

The reachability-emptiness problem is undecidable for PERAs

Proof idea.

By encoding a 2-counter machine into a PERA (using a proof of undecidability for parametric timed automata [ICFEM 2016]) □

Consequence: no hope for exact synthesis

An undecidability result

Theorem (Undecidability)

The reachability-emptiness problem is undecidable for PERAs

Proof idea.

By encoding a 2-counter machine into a PERA (using a proof of undecidability for parametric timed automata [ICFEM 2016]) □

Consequence: no hope for exact synthesis… but

- One can design semi-algorithms
- One can design algorithms synthesizing under- or over-approximations

Outline

Goal

- Algorithms for synthesis in parametric timed formalisms are very expensive
  - No efficient data structures (in contrast to BDDs or DBMs)
    - (Even though parametric DBMs were proposed [Annichini et al., 2000, Hune et al., 2002])
  - Expensive operations on polyhedra (e. g., using PPL [Bagnara et al., 2008])

Goal

- Algorithms for synthesis in parametric timed formalisms are very expensive
  - No efficient data structures (in contrast to BDDs or DBMs)
    - (Even though parametric DBMs were proposed [Annichini et al., 2000, Hune et al., 2002])
  - Expensive operations on polyhedra (e. g., using PPL [Bagnara et al., 2008])

Goal

Design an efficient (semi-)algorithm for the parameter synthesis for PERAs

Goal

- Algorithms for synthesis in parametric timed formalisms are very expensive
  - No efficient data structures (in contrast to BDDs or DBMs)
    - (Even though parametric DBMs were proposed [Annichini et al., 2000, Hune et al., 2002])
  - Expensive operations on polyhedra (e. g., using PPL [Bagnara et al., 2008])

Goal

Design an efficient (semi-)algorithm for the parameter synthesis for PERAs

Idea

Use learning-based techniques to compute an abstraction of the non-parametric components, so as to speed up the verification

Assume-Guarantee Reasoning (AGR)

$$\frac{\begin{array}{rcl} \mathsf{A} \parallel \widetilde{\mathsf{B}} & \models & \varphi \\ \mathsf{B} & \models & \widetilde{\mathsf{B}} \end{array}}{\mathsf{A} \parallel \mathsf{B} \models \varphi}$$

"If A with an abstraction $\widetilde{\mathsf{B}}$ of B satisfy property φ
then A with B satisfy φ"

Assumption and partitioning heuristics

- The system is modeled using a network of PERAs
  - Some components are parametric (i. e., contains parameters)
  - Some components are non-parametric

Assumption and partitioning heuristics

- The system is modeled using a network of PERAs
  - Some components are parametric (i. e., contains parameters)
  - Some components are non-parametric

- Partitioning the system into A ∥ B
  1. If a component has timing parameters, it is collected in group A;
  2. If a component shares common action labels with the property, the component is collected in group A.

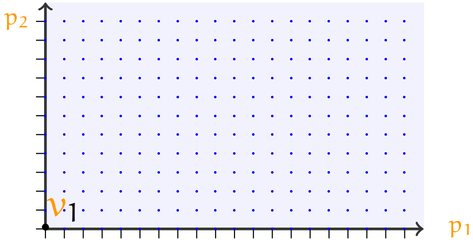
  Other components are collected in group B

Point-based parameter synthesis

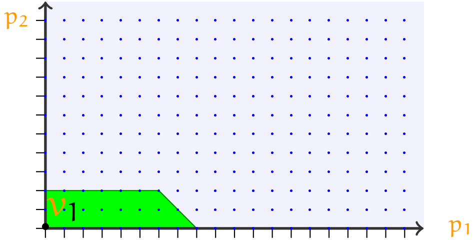
In order to apply (non-parametric) AGR abstractions, we use a point-based method

- Iterate on points (parameter valuations) in a bounded parameter domain
  - e. g., integer points
- For each not point by a parameter constraint, generalize the point using the algorithm PRP
  - PRP: Parametric reachability preservation [NFM 2015]
  - Synthesizes a dense constraint around a point that preserves the reachability of a location
    - If the location is reachable for v, then it is for the constraint
    - and vice-versa
- ... until (at least) a certain set of discrete points is covered
  - e. g., all integers

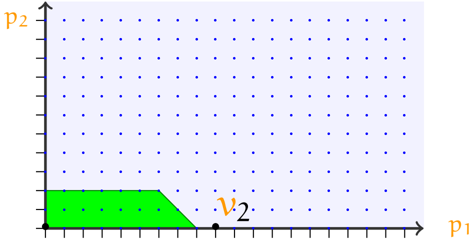
Parametric reachability preservation cartography



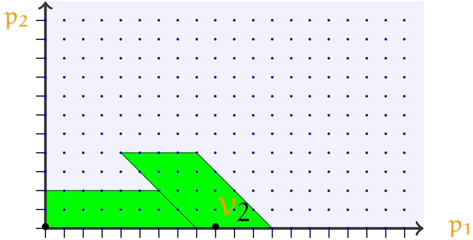
Parametric reachability preservation cartography



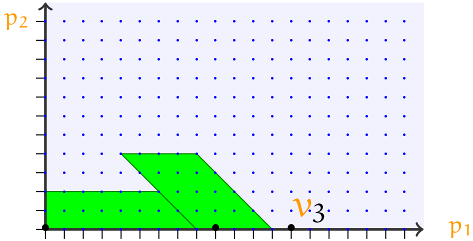
Parametric reachability preservation cartography



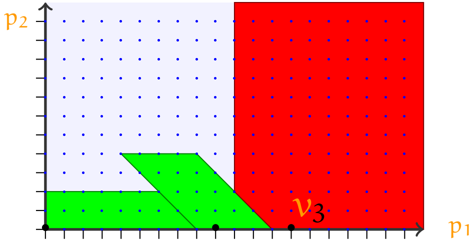
Parametric reachability preservation cartography



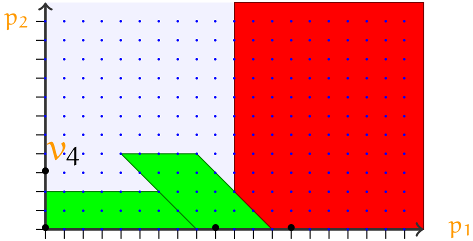
Parametric reachability preservation cartography



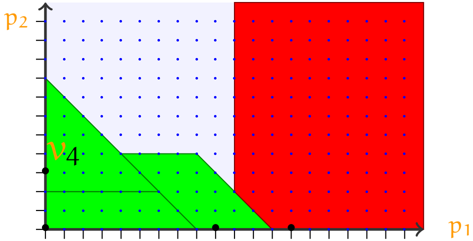
Parametric reachability preservation cartography



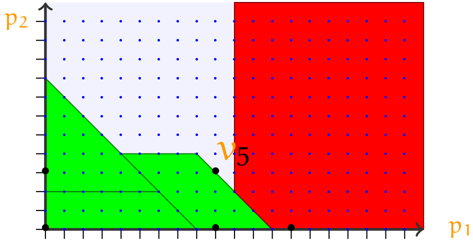
Parametric reachability preservation cartography



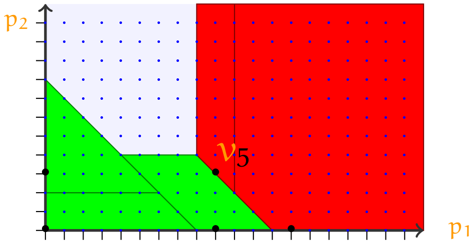
Parametric reachability preservation cartography



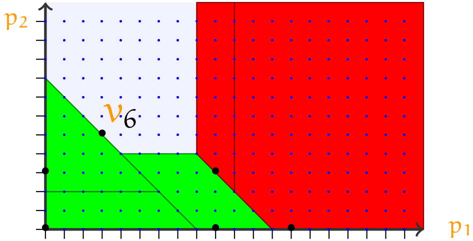
Parametric reachability preservation cartography



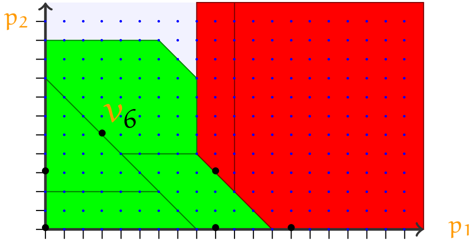
Parametric reachability preservation cartography



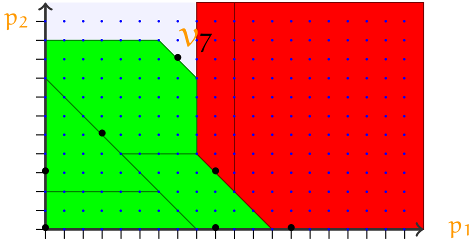
Parametric reachability preservation cartography



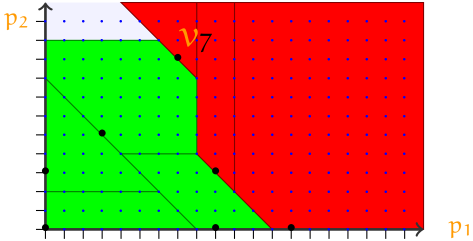
Parametric reachability preservation cartography



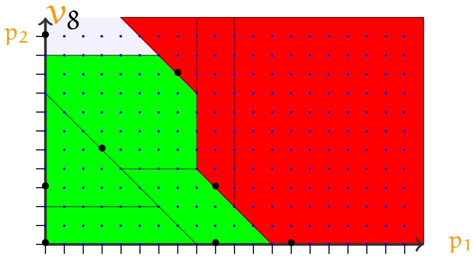
Parametric reachability preservation cartography



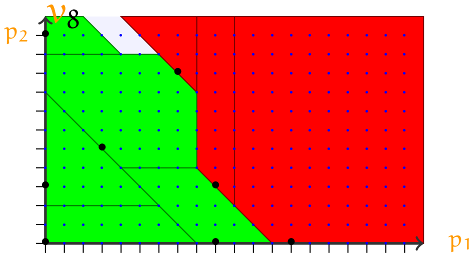
Parametric reachability preservation cartography



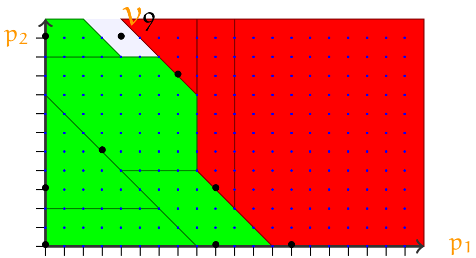
Parametric reachability preservation cartography



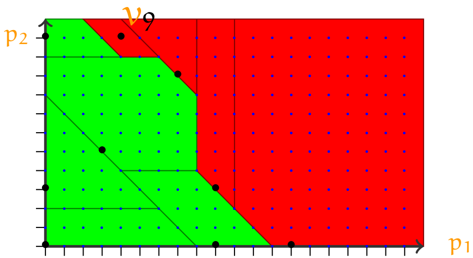
Parametric reachability preservation cartography



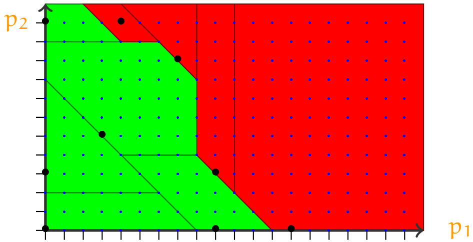
Parametric reachability preservation cartography



Parametric reachability preservation cartography



Parametric reachability preservation cartography



Learning an abstraction: $\mathsf{LearnAbstr}(\mathsf{B}, v(\mathsf{A}), AG\neg L^{☹})$

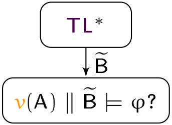
Input: $v(\mathsf{A}) \parallel \mathsf{B}$
Output: an abstraction $\widetilde{\mathsf{B}}$ or a counter-example

TL*

TL*: learning algorithm to compute a candidate abstraction $\widetilde{\mathsf{B}}$ of an ERA B [Lin et al., 2014]

Learning an abstraction: $\mathsf{LearnAbstr}(\mathsf{B}, v(\mathsf{A}), AG\neg L^{☹})$

Input: $v(\mathsf{A}) \parallel \mathsf{B}$
Output: an abstraction $\widetilde{\mathsf{B}}$ or a counter-example



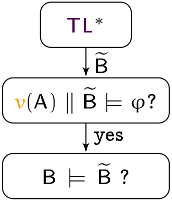
TL*: learning algorithm to compute a candidate abstraction $\widetilde{\mathsf{B}}$ of an ERA B [Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: LearnAbstr$(B, \nu(A), AG\neg L^{☹})$

Input: $\nu(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



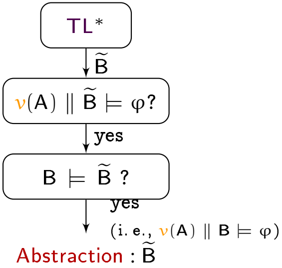
TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B [Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: LearnAbstr(B, ν(A), AG¬L☹)

Input: $\nu(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



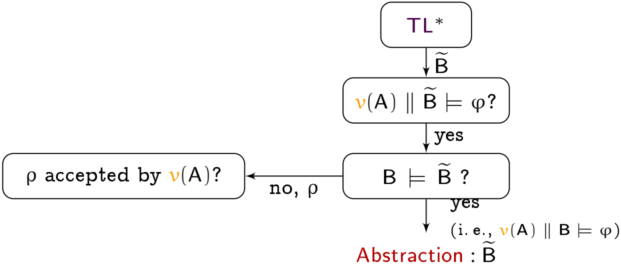
TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B [Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: $\mathsf{LearnAbstr}(B, \nu(A), AG\neg L^{☹})$

Input: $\nu(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



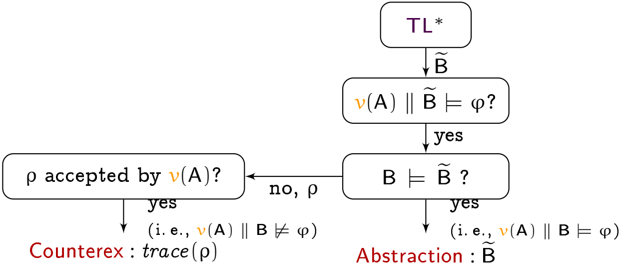
TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B [Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: LearnAbstr$(B, v(A), AG\neg L^{☹})$

Input: $v(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



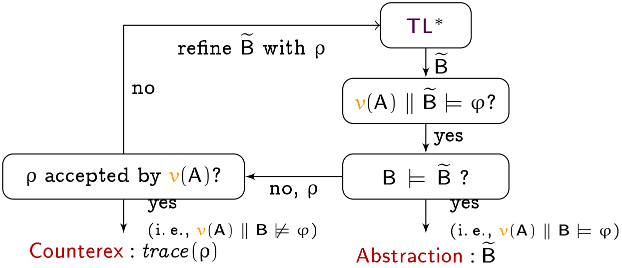
TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B [Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: LearnAbstr$(B, v(A), AG\neg L^{☹})$

Input: $v(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B [Lin et al., 2014]

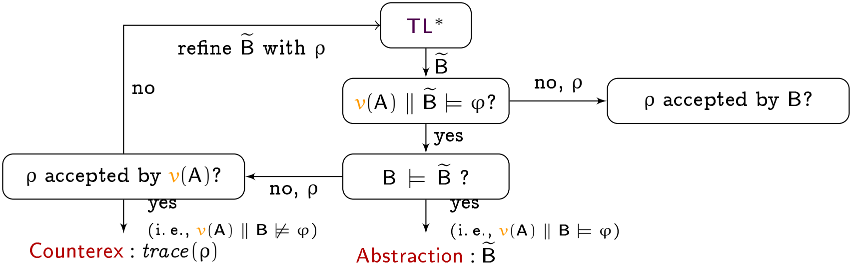
$\models$: can be checked using model checking

Refinement: can be performed using learning

Learning an abstraction: LearnAbstr$(B, \nu(A), AG\neg L^{☹})$

Input: $\nu(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B [Lin et al., 2014]

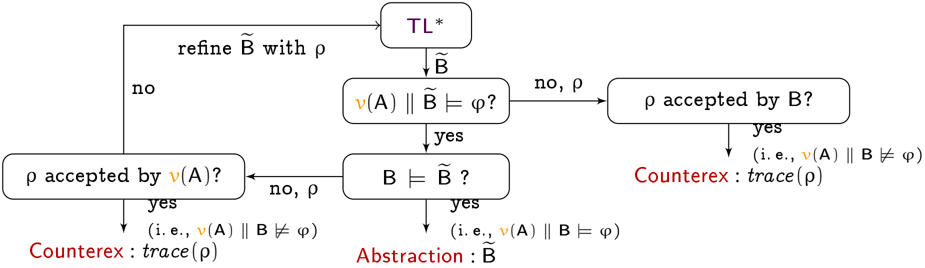
$\models$: can be checked using model checking

Refinement: can be performed using learning

Learning an abstraction: LearnAbstr(B, ν(A), AG¬L☹)

Input: $\nu(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B [Lin et al., 2014]

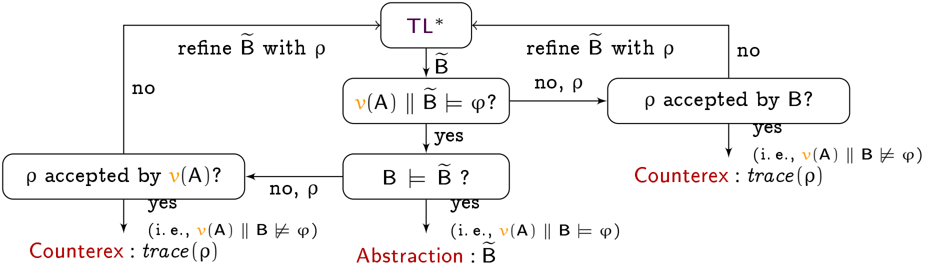
$\models$: can be checked using model checking

Refinement: can be performed using learning

Learning an abstraction: LearnAbstr$(B, \nu(A), AG\neg L^{☹})$

Input: $\nu(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B [Lin et al., 2014]

$\models$: can be checked using model checking

Refinement: can be performed using learning

Replaying a trace

Given a finite trace (i. e., a sequence of actions), we can replay it in the parametric framework

- i. e., find all parameter valuations for which this trace is feasible
- Using a symbolic semantics defined for PERAs (see paper)

☺ Very cheap

Our overall procedure CompSynth

Key ideas:

- Iterate on integer points v
- Try to compute an abstraction $\widetilde{B}$ of the non-parametric component w.r.t. $v(A)$ and φ
  - If succeed, synthesize "similar" valuations using PRP on $A \parallel \widetilde{B}$
  - If fail, synthesize the valuations corresponding to the counterex.

Our overall procedure CompSynth

Key ideas:

- Iterate on integer points v
- Try to compute an abstraction $\widetilde{B}$ of the non-parametric component w.r.t. $v(A)$ and φ
  - If succeed, synthesize "similar" valuations using PRP on $A \parallel \widetilde{B}$
  - If fail, synthesize the valuations corresponding to the counterex.

$K_{bad} \leftarrow \bot$; $K_{good} \leftarrow \bot$
while *there is an integer point not covered by* K_{bad} *or* K_{good} **do**
  Pick such a point v
  switch $\mathsf{LearnAbstr}(B, v(A), AG\neg L^{☹})$ **do**
    case $\mathsf{Abstraction}(\widetilde{B})$
      $K_{good} \leftarrow K_{good} \cup \mathsf{PRP}(A \parallel \widetilde{B}, v, L^{☹})$
    case $\mathsf{Counterex}(\tau)$
      $K_{bad} \leftarrow K_{bad} \cup \mathsf{ReplayTrace}(A \parallel B, \tau)$
return (K_{good}, K_{bad})

Soundness and termination

Proposition (Soundness)

Assume $\mathsf{CompSynth}(\mathsf{A}, \mathsf{B}, \mathrm{L}^{☹})$ *terminates with result* $(\mathsf{K}_{good}, \mathsf{K}_{bad})$.
Then, for all v

1. *if* $v \models \mathsf{K}_{good}$ *then* $v(\mathsf{A} \parallel \mathsf{B})$ *does not reach* $\mathrm{L}^{☹}$;
2. *if* $v \models \mathsf{K}_{bad}$ *then* $v(\mathsf{A} \parallel \mathsf{B})$ *reaches* $\mathrm{L}^{☹}$.

Soundness and termination

Proposition (Soundness)

Assume $\mathsf{CompSynth}(A, B, L^{☹})$ *terminates with result* (K_{good}, K_{bad}). *Then, for all* v

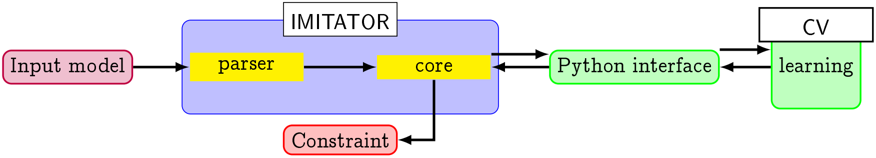
1. *if* $v \models K_{good}$ *then* $v(A \parallel B)$ *does not reach* $L^{☹}$*;*
2. *if* $v \models K_{bad}$ *then* $v(A \parallel B)$ *reaches* $L^{☹}$*.*

Proposition (Integer-completeness)

Assume $\mathsf{CompSynth}(A, B, L^{☹})$ *terminates with result* (K_{good}, K_{bad}). *Then any integer point in the bounded parameter domain is either in* K_{good} *or in* K_{bad}*.*

Outline

Our toolkit



- IMITATOR: state-of-the-art tool for parameter synthesis for parametric timed automata [André et al., 2012]
- CV: (new) prototype compositional verifier of the compositional verification framework for ERAs [Lin et al., 2014]
- interface in Python (about 700 lines)

Experiments

Case study	#A	#X	#P	Spec	EFsynth	PRPC		CompSynth			
						#iter	total	#abs	#c.-ex.	learning	total
FMS-1	6	18	2	1	0.299	2	0.654	1	1	0.074	**0.136**
				2	**0.010**	1	0.372	0	1	0.038	0.046
				3	0.282	1	0.309	1	0	0.090	**0.242**
FMS-2	11	37	2	1	T.O.	-	T.O.	1	1	84.2	**88.9**
				2	T.O.	-	T.O.	1	0	81.4	**85.2**
				3	**0.051**	-	T.O.	0	2	1.10	2.44
				4	**0.062**	-	T.O.	0	1	1.42	1.53
				5	T.O.	-	T.O.	1	0	31.4	**40.8**
				6	T.O.	-	T.O.	1	0	37.2	**42.4**
AIP	11	46	2	1	0.551	-	T.O.	0	1	0.086	**0.114**
				2	2.11	-	T.O.	0	1	1.22	**1.25**
				3	**3.91**	-	T.O.	0	1	8.50	8.54
				4	**0.235**	-	T.O.	1	1	8.39	8.42
				5	T.O.	-	T.O.	1	0	0.394	**0.871**
				6	T.O.	-	T.O.	1	0	5.32	**9.58**
				7	T.O.	-	T.O.	1	0	1.76	**3.19**
				8	T.O.	-	T.O.	1	0	1.13	**4.35**
				9	T.O.	-	T.O.	1	1	0.762	**1.84**
				10	**0.022**	-	T.O.	0	1	0.072	0.094
Fischer-3	5	12	2		**2.76**	4	14.0	0	1	-	T.O.
Fischer-4	6	16	2		T.O.	-	T.O.	0	1	-	T.O.

EFsynth: monolithic reachability synthesis

PRPC: point-based synthesis without abstraction

All data (sources, binaries, models, logs) are available at https://www.imitator.fr/static/FORTE17/

Experiments: scalability

Testing the scalability w.r.t. the size of the parameter domain

Case study	#A	#X	#P	Spec	D_0	CompSynth				
						#abs	#c.-ex.	find next point	learning	total
FMS-2	11	37	2	1	2,500	1	1	0.0	81.0	85.7
					10,000	1	1	0.1	82.5	87.3
					250,000	1	1	2.2	82.0	89.0
					1,000,000	1	1	8.9	83.1	96.7
					25,000,000	1	1	221.2	83.1	309.0
					100,000,000	1	1	888.1	83.5	976.4

Gets slower from 10^6 points

- Reason: use exhaustive enumeration

Experiments: scalability

Testing the scalability w.r.t. the size of the parameter domain

Case study	#A	#X	#P	Spec	D_0	CompSynth				
						#abs	#c.-ex.	find next point	learning	total
FMS-2	11	37	2	1	2,500	1	1	0.0	81.0	85.7
					10,000	1	1	0.1	82.5	87.3
					250,000	1	1	2.2	82.0	89.0
					1,000,000	1	1	8.9	83.1	96.7
					25,000,000	1	1	221.2	83.1	309.0
					100,000,000	1	1	888.1	83.5	976.4

Gets slower from 10^6 points

- Reason: use exhaustive enumeration

Future work: use, e. g., an SMT solver

Outline

Summary

- New method for parameter synthesis for distributed systems, modeled by PERAs
  - PERAs: Strong assumption! Event-recording automata are needed to perform language inclusion in TL^* (undecidable for timed automata)
- Despite undecidability in general, proposed an efficient algorithm
  - Experiments show a dramatic improvement for loosely-synchronized PERAs

Perspectives

- Reuse the distributed point-based synthesis on a cluster
  [A., Coti, Nguyen, ICFEM 2015]
- Beyond PERAs: what can we reuse to perform compositional verification of parametric timed automata?
- Combine our approach with that of [Aştefănoaei et al., 2016]
  - Invariant-based compositional synthesis

Bibliography

References I

Alur, R. and Dill, D. L. (1994).
A theory of timed automata.
Theoretical Computer Science, 126(2):183–235.

Alur, R., Fix, L., and Henzinger, T. A. (1999).
Event-clock automata: A determinizable class of timed automata.
Theoretical Computer Science, 211(1-2):253–273.

Alur, R., Henzinger, T. A., and Vardi, M. Y. (1993).
Parametric real-time reasoning.
In *STOC*, pages 592–601. ACM.

André, É., Coti, C., and Nguyen, H. G. (2015a).
Enhanced distributed behavioral cartography of parametric timed automata.
In *ICFEM*, Lecture Notes in Computer Science. Springer.

André, É., Fribourg, L., Kühne, U., and Soulat, R. (2012).
IMITATOR 2.5: A tool for analyzing robustness in scheduling problems.
In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer.

References II

André, É., Lime, D., and Roux, O. H. (2016).
Decision problems for parametric timed automata.
In *ICFEM*, volume 10009 of *Lecture Notes in Computer Science*, pages 400–416. Springer.

André, É., Lipari, G., Nguyen, H. G., and Sun, Y. (2015b).
Reachability preservation based parameter synthesis for timed automata.
In *NFM*, volume 9058 of *Lecture Notes in Computer Science*, pages 50–65. Springer.

Annichini, A., Asarin, E., and Bouajjani, A. (2000).
Symbolic techniques for parametric reasoning about counter and clock systems.
In *CAV'00*, pages 419–434. Springer-Verlag.

Aştefănoaei, L., Bensalem, S., Bozga, M., Cheng, C., and Ruess, H. (2016).
Compositional parameter synthesis.
In *FM*, volume 9995 of *Lecture Notes in Computer Science*, pages 60–68.

Bagnara, R., Hill, P. M., and Zaffanella, E. (2008).
The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems.
Science of Computer Programming, 72(1–2):3–21.

References III

Hune, T., Romijn, J., Stoelinga, M., and Vaandrager, F. W. (2002).
Linear parametric model checking of timed automata.
Journal of Logic and Algebraic Programming, 52-53:183–220.

Larsen, K. G., Pettersson, P., and Yi, W. (1997).
UPPAAL in a nutshell.
International Journal on Software Tools for Technology Transfer, 1(1-2):134–152.

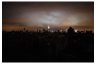
Lin, S.-W., André, É., Liu, Y., Sun, J., and Dong, J. S. (2014).
Learning assumptions for compositional verification of timed systems.
Transactions on Software Engineering, 40(2):137–153.

Markey, N. (2011).
Robustness in real-time systems.
In *SIES*, pages 28–34. IEEE Computer Society Press.

Sun, J., Liu, Y., Dong, J. S., and Pang, J. (2009).
PAT: Towards flexible verification under fairness.
In *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer.

Additional explanation

Explanation for the 4 pictures in the beginning



Allusion to the Northeast blackout (USA, 2003)
Computer bug
Consequences: 11 fatalities, huge cost
(Picture actually from the Sandy Hurricane, 2012)

Error screen on the earliest versions of Macintosh

Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
No fatalities
Computer bug: inaccurate finite element analysis modeling
(Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
28 fatalities, hundreds of injured
Computer bug: software error (clock drift)
(Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)

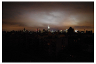
Experiments: partitioning

Test various partitioning heuristics

Case study	#A	#X	#P	Spec	Partitioning A	Partitioning B	CompSynth #abs	CompSynth #c.-ex.	CompSynth learning	CompSynth total
FMS-1	6	18	2	1	CM	R_1R_2A	1	1	1.071	1.137
					CMR_1	R_2A	1	1	0.077	**0.148**
					CMR_2	R_1A	1	1	5.152	5.406
					CMA	R_1R_2	1	1	5.663	5.980
					CMR_1R_2	A	1	1	0.123	0.290
					CMR_1A	R_2	1	1	0.119	0.360
					CMR_2A	R_1	1	1	6.150	6.690
				2	CM	R_1R_2A	0	1	0.133	0.149
					CMR_1	R_2A	0	2	0.077	0.123
					CMR_2	R_1A	0	1	0.040	0.056
					CMA	R_1R_2	0	1	0.824	0.842
					CMR_1R_2	A	0	1	0.034	**0.044**
					CMR_1A	R_2	0	2	0.096	0.144
					CMR_2A	R_1	0	1	0.042	0.051
				3	CM	R_1R_2A	1	0	0.211	0.270
					CMR_1	R_2A	1	0	0.082	**0.186**
					CMR_2	R_1A	1	0	1.094	1.208
					CMA	R_1R_2	1	0	0.729	0.881
					CMR_1R_2	A	1	0	0.119	0.279
					CMR_1A	R_2	1	0	0.314	0.634
					CMR_2A	R_1	1	0	0.104	0.257

Licensing

Source of the graphics used I



Title: Hurricane Sandy Blackout New York Skyline
Author: David Shankbone
Source: https://commons.wikimedia.org/wiki/File:Hurricane_Sandy_Blackout_New_York_Skyline.JPG
License: CC BY 3.0

Title: Sad mac
Author: Przemub
Source: https://commons.wikimedia.org/wiki/File:Sad_mac.png
License: Public domain

Title: Deepwater Horizon Offshore Drilling Platform on Fire
Author: ideum
Source: https://secure.flickr.com/photos/ideum/4711481781/
License: CC BY-SA 2.0

Title: DA-SC-88-01663
Author: imcomkorea
Source: https://secure.flickr.com/photos/imcomkorea/3017886760/
License: CC BY-NC-ND 2.0

Source of the graphics used II

Title: Smiley green alien big eyes (aaah)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

Title: Smiley green alien big eyes (cry)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

License of this document