

ICTAC '09

IMITATOR: A Tool for Synthesizing Constraints on Timing Bounds of Timed Automata

Étienne ANDRÉ

Laboratoire Spécification et Vérification
LSV, ENS de Cachan & CNRS

Context: Real-Time Concurrent Systems

- Verification of safety property: ensure the **absence** of any **bad behavior** (reachability property)
- A well-known method: CEGAR (Counter-Example Guided Abstraction Refinement [CGJ⁺00])
 - ▶ They use (repeatedly) an example of **bad behavior** in order to **refine** the model of the system
- We present here a **generalization** method
 - ▶ Implementation of the **inverse method** [ACEF09]
 - ▶ We use a given example of **good behavior** in order to **generalize** the model of the system

Outline

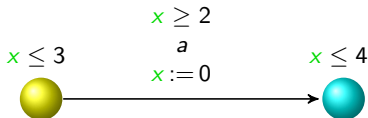
- 1 The Modeling Framework of Parametric Timed Automata
- 2 Presentation of IMITATOR
 - Overview
 - A Motivating Example
 - The General Idea
 - Application to the Example
- 3 Implementation and Case Studies
- 4 Conclusion and Future Works

Outline

- 1 The Modeling Framework of Parametric Timed Automata
- 2 Presentation of IMITATOR
 - Overview
 - A Motivating Example
 - The General Idea
 - Application to the Example
- 3 Implementation and Case Studies
- 4 Conclusion and Future Works

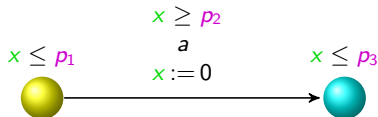
Timed Automata

- Finite state automata (sets of **locations** and **labeled transitions**) augmented with
 - ▶ A set X of **clocks** (i.e., real-valued variables evolving linearly at the same rate)
- Operations
 - ▶ Location **invariant**: property to be verified by the **clocks** to stay at a location
 - ▶ Transition **guard**: property to be verified by the **clocks** to enable a transition
 - ▶ Clock **reset**: clocks can be set to 0 at each transition



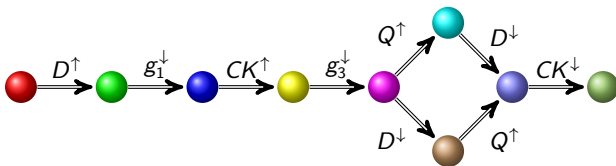
Parametric Timed Automata (PTA)

- Finite state automata (sets of **locations** and **labeled transitions**) augmented with
 - ▶ A set X of **clocks** (i.e., real-valued variables evolving linearly at the same rate)
 - ▶ A set P of **parameters** (i.e., unknown constants), used in guards and invariants
- Operations
 - ▶ Location **invariant**: property to be verified by the **clocks** and the **parameters** to stay at a location
 - ▶ Transition **guard**: property to be verified by the **clocks** and the **parameters** to enable a transition
 - ▶ Clock **reset**: clocks can be set to 0 at each transition



States and Traces

- **Symbolic state** of a PTA: couple (q, C) , where
 - ▶ q is a location,
 - ▶ C is a **constraint** (conjunction of inequalities) over the **parameters**
- **Trace** (or run) over a PTA: finite alternating sequence of **locations** and transitions



Outline

- 1 The Modeling Framework of Parametric Timed Automata
- 2 **Presentation of IMITATOR**
 - Overview
 - A Motivating Example
 - The General Idea
 - Application to the Example
- 3 Implementation and Case Studies
- 4 Conclusion and Future Works

Inputs and Outputs (1/2)



Inputs and Outputs (2/2)

- Input

- ▶ A PTA \mathcal{A}
- ▶ A reference instantiation π_0 of all the parameters of \mathcal{A}
 - ★ Exemplifying a good behavior
(all traces under π_0 represent good behaviors)

π_0

Inputs and Outputs (2/2)

- **Input**

- ▶ A PTA \mathcal{A}
- ▶ A **reference instantiation** π_0 of all the parameters of \mathcal{A}
 - ★ Exemplifying a good behavior
(all traces under π_0 represent good behaviors)

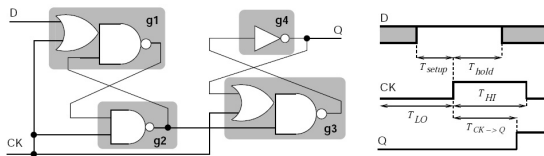
- **Output:** generalization

- ▶ A **constraint** K_0 on the parameters such that
 - ★ $\pi_0 \models K_0$
 - ★ For all instantiation $\pi \models K_0$, the set of traces under π is the same as the set of traces under π_0



An Example of Circuit (1/2)

- Memory circuit [CC07]



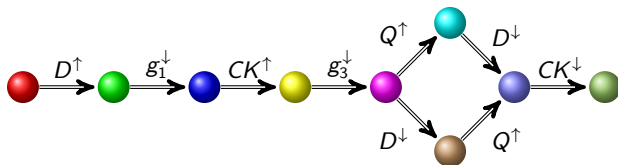
- 4 elements: G_1, G_2, G_3, G_4
- 2 input signals (D and CK), 1 output signal (Q)
- 4 internal signals: g_1, g_2, g_3, g_4 (output of each element)
- Timed parameters of the system
 - Traversal delays of the gates by the electric current
 - ★ Parametric interval; example for element g_1 : $[\delta_1^-, \delta_1^+]$
 - Stabilization time of input signal D
 - ★ T_{Setup}, T_{Hold}
 - CK low and high durations
 - ★ T_{LO}, T_{HI}

An Example of Circuit (2/2)

- We are given an **instantiation of the parameters**

$$\begin{array}{cccc}
 T_{HI} = 20 & T_{LO} = 15 & T_{Setup} = 10 & T_{Hold} = 15 \\
 \delta_1^- = 1 & \delta_1^+ = 1 & \delta_3^- = 5 & \delta_3^+ = 6 \\
 \delta_2^- = 8 & \delta_2^+ = 10 & \delta_4^- = 3 & \delta_4^+ = 5
 \end{array}$$

- ▶ This instantiation point guarantees some **(good) behavior**



- We are looking for **other instantiations** of the parameters leading to the **same (good) behavior**

The General Idea of IMITATOR

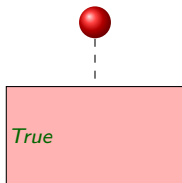
Start with $K_0 = True$

- 1 Compute the set S of reachable symbolic states under K_0
- 2 Refine K_0 by removing a π_0 -incompatible state from S
 - ▶ Select a π_0 -incompatible state (q, C) within S (i.e., $\pi_0 \not\models C$)
 - ▶ Select a π_0 -incompatible inequality J within C (i.e., $\pi_0 \not\models J$)
 - ▶ Add $\neg J$ to K_0
- 3 Go to (1)

Application to the Memory Circuit Example

 $\pi_0 :$

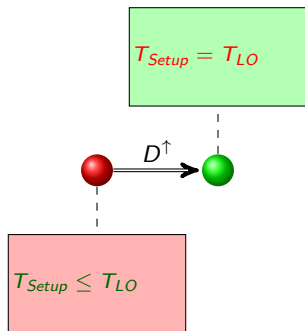
$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

 $K_0 = True$


Application to the Memory Circuit Example

 $\pi_0 :$

$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

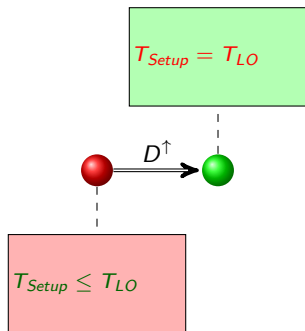
 $K_0 = True$


Application to the Memory Circuit Example

 $\pi_0 :$

$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

$$K_0 = T_{Setup} < T_{LO}$$



Application to the Memory Circuit Example

 $\pi_0 :$

$$\delta_1^- = 1 \quad \delta_1^+ = 1 \quad T_{HI} = 20$$

$$\delta_2^- = 8 \quad \delta_2^+ = 10 \quad T_{LO} = 15$$

$$\delta_3^- = 5 \quad \delta_3^+ = 6 \quad T_{Setup} = 10$$

$$\delta_4^- = 3 \quad \delta_4^+ = 5 \quad T_{Hold} = 15$$

$$K_0 = T_{Setup} < T_{LO}$$



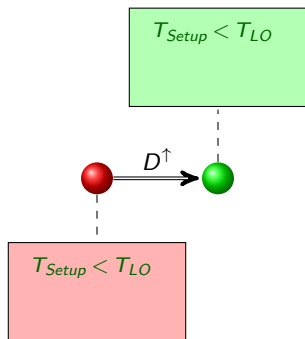
$$T_{Setup} < T_{LO}$$

Application to the Memory Circuit Example

 $\pi_0 :$

$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

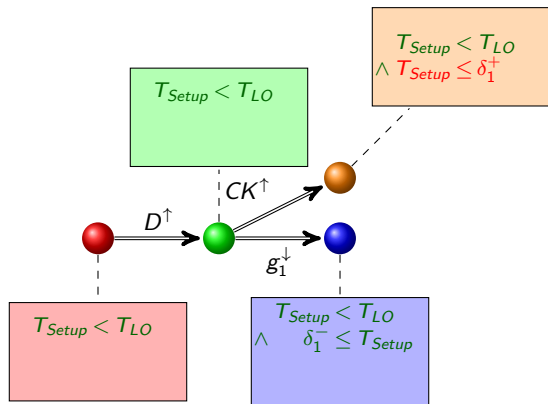
$$K_0 = T_{Setup} < T_{LO}$$



Application to the Memory Circuit Example

$\pi_0 :$		
$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

$$K_0 = T_{Setup} < T_{LO}$$



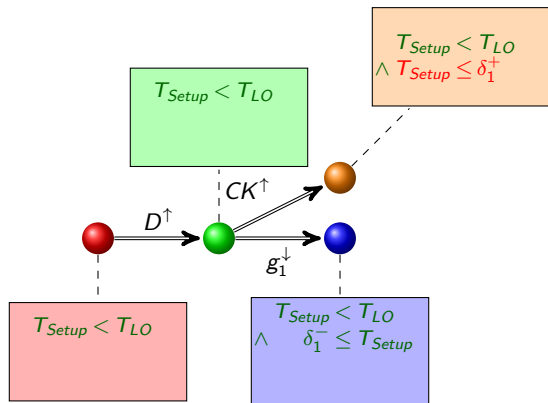
Application to the Memory Circuit Example

$\pi_0 :$		
$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

$$K_0 =$$

$$T_{Setup} < T_{LO}$$

$$\wedge T_{Setup} > \delta_1^+$$



Application to the Memory Circuit Example

 $\pi_0 :$

$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

 $K_0 =$

$$T_{Setup} < T_{LO}$$

$$\wedge T_{Setup} > \delta_1^+$$



$$T_{Setup} < T_{LO}$$

$$\wedge T_{Setup} > \delta_1^+$$

Application to the Memory Circuit Example

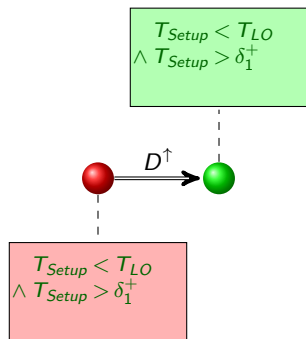
 $\pi_0 :$

$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

 $K_0 =$

$$T_{Setup} < T_{LO}$$

$$\wedge T_{Setup} > \delta_1^+$$



Application to the Memory Circuit Example

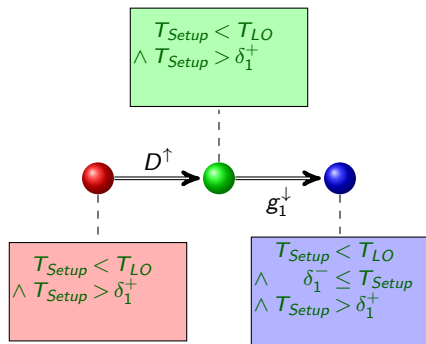
 $\pi_0 :$

$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

 $K_0 =$

$$T_{Setup} < T_{LO}$$

$$\wedge T_{Setup} > \delta_1^+$$



Application to the Memory Circuit Example

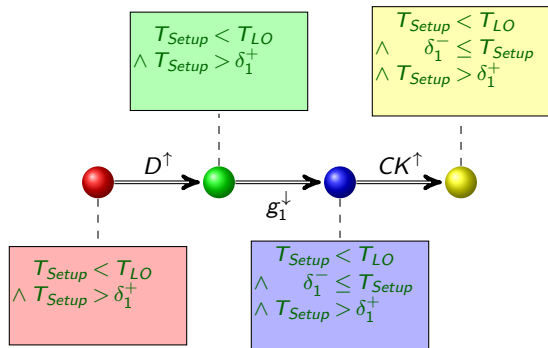
 $\pi_0 :$

$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

 $K_0 =$

$$T_{Setup} < T_{LO}$$

$$\wedge T_{Setup} > \delta_1^+$$



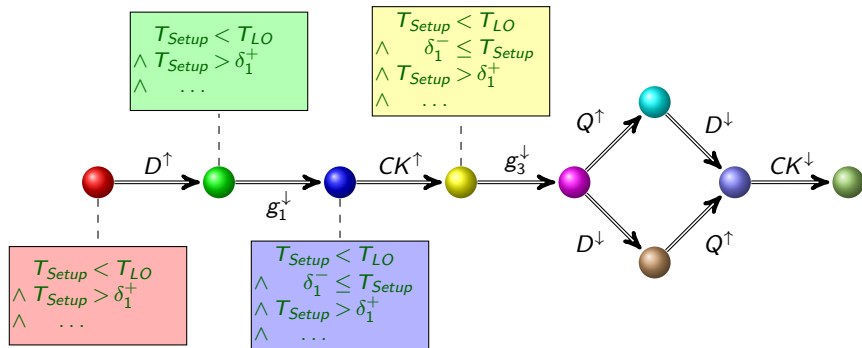
Application to the Memory Circuit Example

 $\pi_0 :$

$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

 $K_0 =$

$$\begin{aligned}
 & T_{Setup} < T_{LO} \quad \wedge \quad \delta_3^+ + \delta_4^+ < T_{HI} \\
 & \wedge T_{Setup} > \delta_1^+ \quad \wedge \quad \delta_3^+ + \delta_4^+ \geq T_{Hold} \\
 & \wedge \quad \delta_3^+ < T_{Hold} \quad \wedge \quad \delta_3^- + \delta_4^- \leq T_{Hold} \\
 & \wedge \quad \delta_1^- > 0
 \end{aligned}$$



Outline

- 1 The Modeling Framework of Parametric Timed Automata
- 2 Presentation of IMITATOR
 - Overview
 - A Motivating Example
 - The General Idea
 - Application to the Example
- 3 Implementation and Case Studies
- 4 Conclusion and Future Works

Implementation

- IMITATOR: program written in Python
 - ▶ IMITATOR: “Inverse Method for Inferring Time Abstract Behavior”
 - ▶ 1500 lines of code
 - ▶ 4 man-months of work
 - ▶ Calls the parametric model checker HYTECH [HHWT95]
 - ★ Tool written in C
 - ★ Used by IMITATOR for the computation of the *Post* operation
- IMITATOR is available on its Web page
 - ▶ <http://www.lsv.ens-cachan.fr/~andre/IMITATOR>

Case Studies

- Some real cases treated
 - ▶ SPSMALL [CEFX09]: memory circuit (ST-Microelectronics)
 - ★ Allow to optimize input timing bounds
 - ▶ SIMOP [AAC⁺09]: model of manufacturing system with sensors and controllers communicating through a network
 - ★ Allow to define zones of good behavior
- Computation times of various case studies [AEF09]
 - ▶ Experiences conducted on an Intel Quad Core 3 Ghz with 3.2 Gb

Example	# of PTAs	loc. per PTA	# of clocks	# of param.	# of iter.	Post*	K ₀	CPU time
Flip-flop [CC07]	5	[4, 16]	5	12	8	11	7	2 s
RCP [SS01]	5	[6, 11]	6	5	18	154	2	70 s
CSMA/CD [NSY92]	3	[6, 7]	4	3	21	294	3	108 s
SPSMALL [CEFX09]	10	[3, 8]	10	22	31	31	23	78 min
SIMOP [AAC ⁺ 09]	5	[6, 16]	9	16	51	848	7	419 min

Outline

- 1 The Modeling Framework of Parametric Timed Automata
- 2 Presentation of IMITATOR
 - Overview
 - A Motivating Example
 - The General Idea
 - Application to the Example
- 3 Implementation and Case Studies
- 4 Conclusion and Future Works

Final Remarks

- Inverse method implemented in IMITATOR
 - ▶ Modeling of a system with parametric timed automata
 - ▶ Starting with an instantiation point π_0 of the system, IMITATOR generates a constraint K_0 on the parameters guaranteeing the same set of traces

- Advantages
 - ▶ Sufficient termination conditions
 - ▶ Useful to optimize timing bounds of systems
 - ▶ Powerful even on fully parameterized big systems
 - ★ Can handle dozens of parameters

- Drawbacks
 - ▶ The zone (set of points) generated by the constraint is rather small compared to exhaustive point by point methods
 - ▶ The generated constraint is not maximal: it is possible to find valuations $\pi \not\models K_0$ s.t. the set of traces under π and π_0 are the same

Future Works

- Improve the constraint K_0
 - ▶ Goal: generate a maximal constraint K_0
 - ★ The maximal K_0 may exist under a disjunctive form only
 - ▶ Use IMITATOR in an incremental way
 - ★ Given a constraint K generated by IMITATOR, run IMITATOR again on a new point $\pi \not\models K$
- Increase the scalability
 - ▶ Write an *ad hoc* tool instead of using HYTECH
 - ▶ Use a library of polyhedra

References I



Saïd Amari, Étienne André, Thomas Chatain, Olivier De Smet, Bruno Denis, Emmanuelle Encrenaz, Laurent Fribourg, and Sylvain Ruel.

Timed analysis of networked automation systems combining simulation and parametric model checking.

Research Report LSV-09-14, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2009.

SIMOP Research Report. 49 pages.



Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg.
An inverse method for parametric timed automata.

International Journal of Foundations of Computer Science, 2009.

To appear.



É. André, E. Encrenaz, and L. Fribourg.

Synthesizing parametric constraints on various case studies using IMITATOR.

Research report, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2009.








R. Clarisó and J. Cortadella.

The octahedron abstract domain.

Sci. Comput. Program., 64(1):115–139, 2007.

References II

-  R. Chevallier, E. Encrenaz, L. Fribourg, and W. Xu.
Timed verification of the generic architecture of a memory circuit using parametric timed automata.
Formal Methods in System Design, 34(1):59–81, February 2009.
-  E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith.
Counterexample-guided abstraction refinement.
In *CAV '00*, pages 154–169. Springer-Verlag, 2000.
-  T. A. Henzinger, P. Ho, and H. Wong-Toi.
A user guide to HYTECH.
In *TACAS*, pages 41–71, 1995.
-  X. Nicollin, J. Sifakis, and S. Yovine.
Compiling real-time specifications into extended automata.
IEEE Trans. on Software Engineering, 18:794–804, 1992.
-  D. Simons and M. Stoelinga.
Mechanical verification of the IEEE 1394a Root Contention Protocol using UPPAAL2k.
International Journal on Software Tools for Technology Transfer, 3(4):469–485, 2001.