

Séminaire MeFoSyLoMa
Vendredi 22 janvier 2010

Synthesis of timing parameters in timed automata for the verification of hardware components

Étienne ANDRÉ

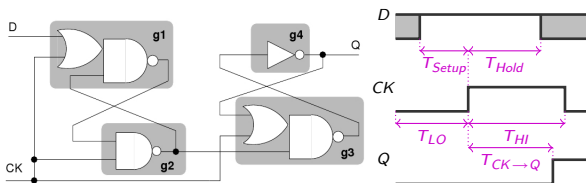
Laboratoire Spécification et Vérification
LSV, ENS de Cachan & CNRS, France

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 generate a counter-example in order to **refine** the model of the system, until there is no more counter-example
- We present here a **generalization** method [ACEF09]
 - ▶ We use a given example of **good behavior** in order to **relax** the timing bounds of the system

An Example of Circuit (1/2)

- Memory circuit [CC07]



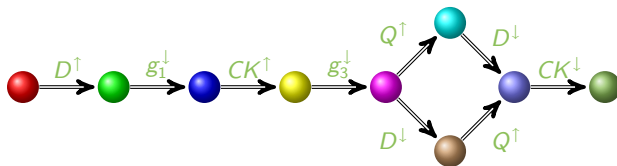
- ▶ 4 elements: G_1 , G_2 , G_3 , G_4
 - ▶ 2 input signals (D and CK), 1 output signal (Q)
 - ▶ 4 internal signals: g_1 , g_2 , g_3 , g_4 (output of each element)
- Timed parameters of the system
 - ▶ Traversal delays of the gates by the electric current
 - ★ Parametric interval; example for element 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 suppose 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 a **good behavior**:
 - ★ Both Q^\uparrow and CK^\downarrow occur
 - ★ Q^\uparrow occurs before CK^\downarrow



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

Outline

- 1 The Modeling Framework of Parametric Timed Automata
- 2 The Inverse Method
 - The General Idea
 - Application to the Example
 - Implementation and Case Studies
 - Discussion
- 3 Extension: Behavioral Cartography
- 4 Final Remarks

Outline

- 1 The Modeling Framework of Parametric Timed Automata
- 2 The Inverse Method
 - The General Idea
 - Application to the Example
 - Implementation and Case Studies
 - Discussion
- 3 Extension: Behavioral Cartography
- 4 Final Remarks

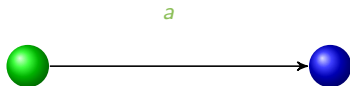
Timed Automaton

- Finite state automaton (sets of *locations*)



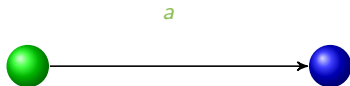
Timed Automaton

- Finite state automaton (sets of **locations** and **actions**)



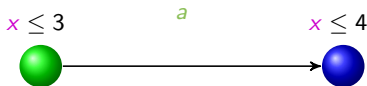
Timed Automaton

- Finite state automaton (sets of **locations** and **actions**) augmented with
 - ▶ A set X of **clocks** (i.e., real-valued variables evolving linearly at the same rate)



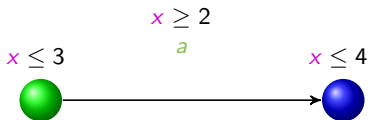
Timed Automaton

- Finite state automaton (sets of **locations** and **actions**) augmented with
 - ▶ A set X of **clocks** (i.e., real-valued variables evolving linearly at the same rate)
- Features
 - ▶ Location **invariant**: property to be verified by the **clocks** to stay at a location



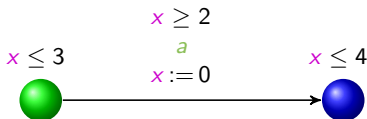
Timed Automaton

- Finite state automaton (sets of **locations** and **actions**) augmented with
 - ▶ A set X of **clocks** (i.e., real-valued variables evolving linearly at the same rate)
- Features
 - ▶ 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



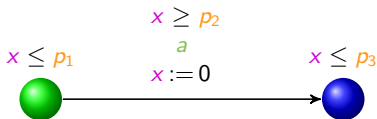
Timed Automaton

- Finite state automaton (sets of **locations** and **actions**) augmented with
 - ▶ A set X of **clocks** (i.e., real-valued variables evolving linearly at the same rate)
- Features
 - ▶ 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



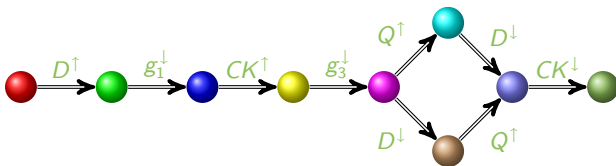
Parametric Timed Automaton (PTA)

- Finite state automaton (sets of **locations** and **actions**) 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
- Features
 - ▶ 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

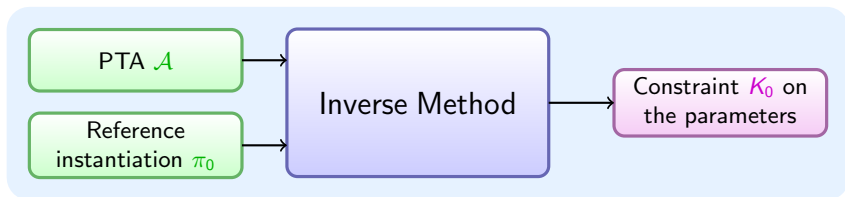
- **Symbolic state** of a PTA: couple (q, C) , where
 - ▶ q is a location,
 - ▶ C is a **constraint** (conjunction of inequalities) over the **parameters**
- **Trace** (time-abstract run) over a PTA: finite alternating sequence of **locations** and **actions**



Outline

- 1 The Modeling Framework of Parametric Timed Automata
- 2 The Inverse Method**
 - The General Idea
 - Application to the Example
 - Implementation and Case Studies
 - Discussion
- 3 Extension: Behavioral Cartography
- 4 Final Remarks

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 correspond to 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 correspond to 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



The General Idea of Our Method

Start with $K_0 = \text{True}$

- 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)

Until fix point (no more π_0 -incompatible states in S)

Correctness and Termination

Theorem (Correctness)

Suppose that $\text{InverseMethod}(\mathcal{A}, \pi_0)$ terminates with output K_0 . Then, we have:

- 1 $\pi_0 \models K_0$, and
- 2 for all $\pi \models K_0$, the sets of traces of $\mathcal{A}[\pi_0]$ and $\mathcal{A}[\pi]$ are equal.

Proposition (Termination)

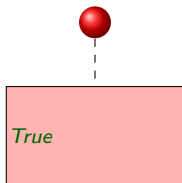
The algorithm terminates if the set of traces of $\mathcal{A}[\pi_0]$ contains no cyclic trace (trace passing twice by the same location).

- The termination can be shown in more cases
- In practice, the algorithm terminates for most of our case studies

Application to the Memory Circuit Example

 $\pi_0 :$

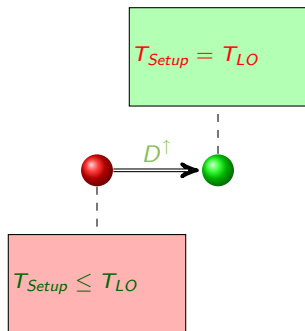
$\delta_1^- = 1$	$\delta_1^+ = 1$	$T_{HI} = 20$
$\delta_2^- = 8$	$\delta_2^+ = 10$	$T_{LO} = 15$
$\delta_3^- = 5$	$\delta_3^+ = 6$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 5$	$T_{Hold} = 15$

 $K_0 = True$


Application to the Memory Circuit Example

 $\pi_0 :$

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

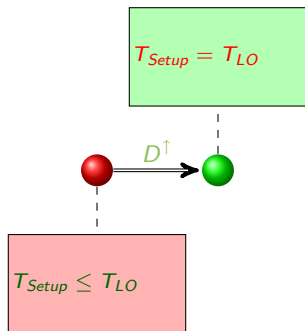
 $K_0 = True$


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

 $\pi_0 :$

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

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



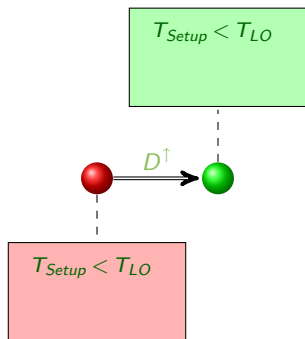
$$T_{Setup} < T_{LO}$$

Application to the Memory Circuit Example

 $\pi_0 :$

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

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

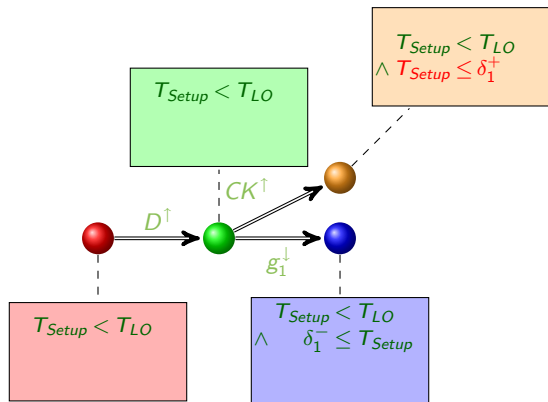


Application to the Memory Circuit Example

 $\pi_0 :$

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

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



Application to the Memory Circuit Example

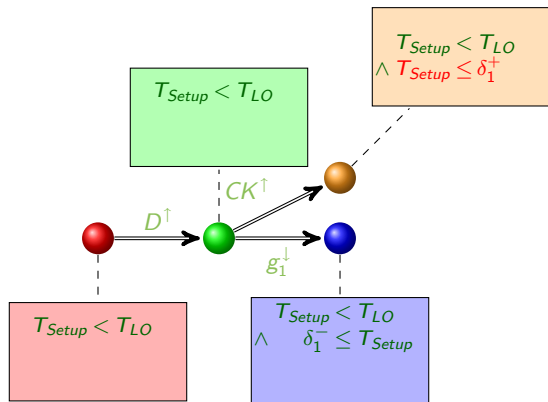
 $\pi_0 :$

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

 $K_0 =$

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

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



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

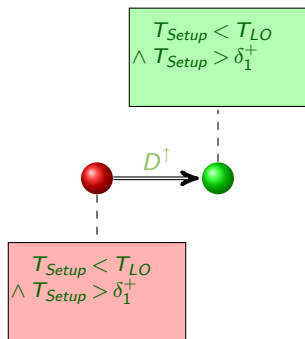
 $\pi_0 :$

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

 $K_0 =$

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

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



Application to the Memory Circuit Example

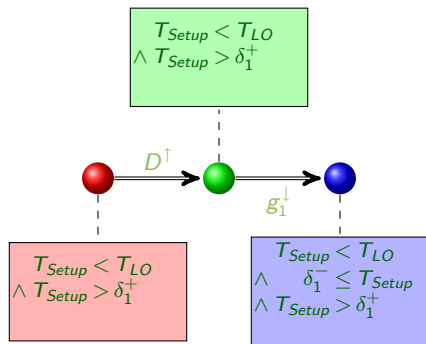
 $\pi_0 :$

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

 $K_0 =$

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

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



Application to the Memory Circuit Example

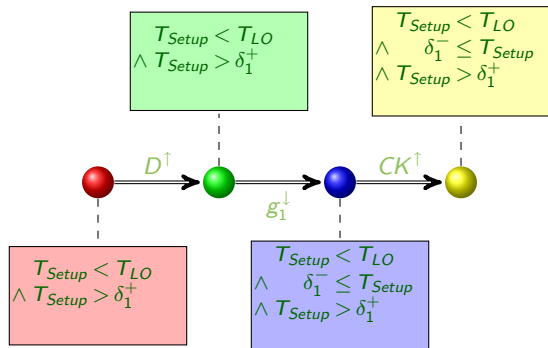
 $\pi_0 :$

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

 $K_0 =$

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

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



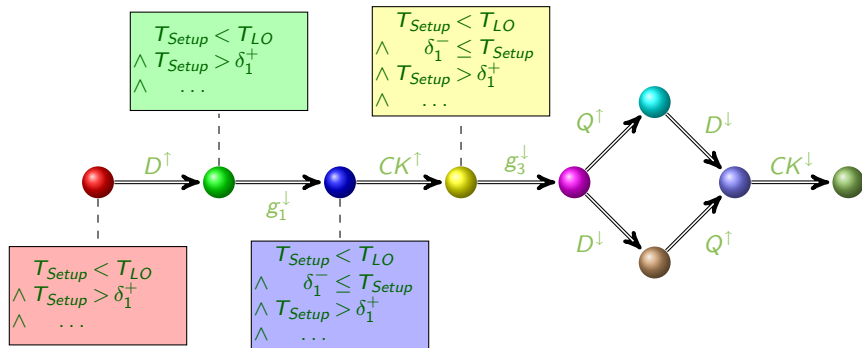
Application to the Memory Circuit Example

 $\pi_0 :$

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

 $K_0 =$

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



Implementation

- IMITATOR [And09]
 - ▶ IMITATOR: “Inverse Method for Inferring Time Abstract Behavior”
 - ▶ 1500 lines of code
 - ▶ 4 man-months of work
 - ▶ Program written in Python
 - ▶ Calls the parametric model checker HYTECH
 - ★ 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 [ACD⁺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
CSMA/CD [KNSW07, wp]	3	[3, 8]	3	3	17	218	3	44 s
RCP [SS01]	5	[6, 11]	6	5	18	154	2	70 s
SPSMALL [CEFX09]	10	[3, 8]	10	22	31	31	23	78 min
SIMOP [ACD ⁺ 09]	5	[6, 16]	9	16	51	848	7	419 min

Implementation: Ongoing Work

- New: **IMITATOR II**
 - ▶ Brand new version of IMITATOR
 - ▶ 6000 lines of code
 - ▶ Program written in **OCaml**
 - ▶ Development in progress, but results already available
- IMITATOR II is available on its Web page
 - ▶ <http://www.lsv.ens-cachan.fr/~andre/IMITATOR2>

Advantages and Drawbacks of the Inverse Method

• 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 **minimal**: 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

Outline

- 1 The Modeling Framework of Parametric Timed Automata
- 2 The Inverse Method
 - The General Idea
 - Application to the Example
 - Implementation and Case Studies
 - Discussion
- 3 Extension: Behavioral Cartography
- 4 Final Remarks

Motivation

- Goal: Find the **maximal set of points** (valuations of the parameters) corresponding to a **good behavior**
- Drawbacks of the inverse method
 - ▶ The generated set of points is not **maximal**
 - ▶ There are good points which correspond to a different behavior from π_0
 - ▶ The maximal set of points is generally not convex
- Idea: Iterate the inverse method for all the integer points of a given rectangle
 - ▶ \rightsquigarrow **behavioral cartography** of the parameter space

The Behavioral Cartography Algorithm

- Principle of the algorithm:
 - ▶ Start with an interval of values V_0 for the parameters
 - ▶ Call the *InverseMethod* on a valuation randomly selected within V_0
 - ▶ Stop when all the integer points within V_0 are covered

Algorithm 2: Behavioral Cartography Algorithm

input : A PTA \mathcal{A} , a rectangle V_0 (interval for the parameters)

output: *Cover*: set of polyhedra (initially empty)

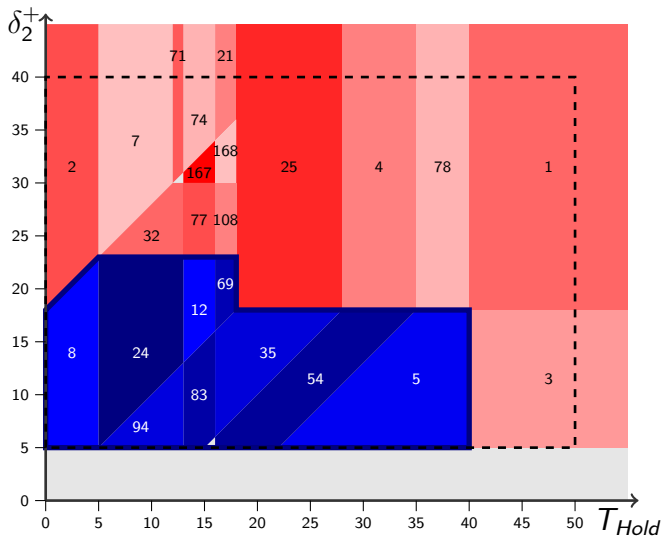
```

1 repeat
2   | select randomly an integer point  $\pi \in V_0$ ;
3   | if  $\pi \notin \textit{Cover}$  then
4   |   |  $\textit{Cover} \leftarrow \textit{Cover} \cup \textit{InverseMethod}(\mathcal{A}, \pi)$ ;
5 until Cover contains all the integer points of the rectangle;
```

Application to our Example of Memory Circuit

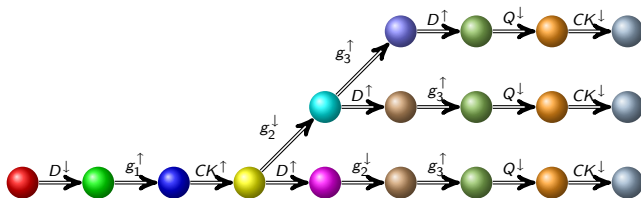
- We consider only parameters T_{Hold} and δ_2^+
 - ▶ The other parameters are instantiated
- Goal: Perform the behavioral cartography of the memory circuit according to T_{Hold} and δ_2^+
- Use of an extension of IMITATOR II

Behavioral Cartography of the Memory Circuit

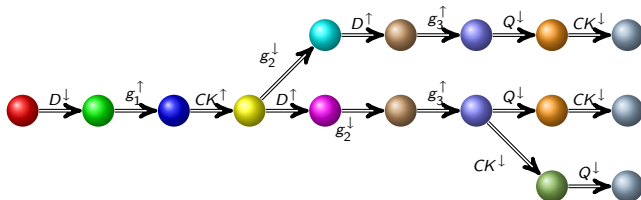


Example of good and bad behaviors (1/2)

- Zone 12 (good behavior)

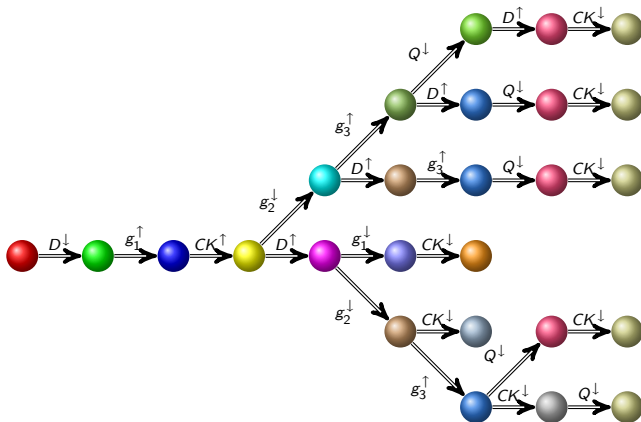


- Zone 32 (bad behavior)



Example of good and bad behaviors (2/2)

- Zone 21 (bad behavior)



Behavioral Cartography of the Memory Circuit: Remarks

- Remarks on the cartography
 - ▶ All the integer points are covered (from the algorithm)
 - ▶ Most of the real-valued part of the space within V_0 is also covered
 - ★ Except 2 triangles containing no integer point
 - ▶ All the (real-valued) space outside V_0 is also covered
- Partition into a good and a bad zone
 - ▶ According to the shape of the trace, we can partition the tiles into good and bad ones
 - ▶ We can generate the good zone (in blue), which is not convex: this zone corresponds to all the good values for δ_2^+ and T_{Hold}

Outline

- 1 The Modeling Framework of Parametric Timed Automata
- 2 The Inverse Method
 - The General Idea
 - Application to the Example
 - Implementation and Case Studies
 - Discussion
- 3 Extension: Behavioral Cartography
- 4 Final Remarks

Summary

- Algorithm *InverseMethod*
 - ▶ 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
 - ▶ Implementations: IMITATOR and IMITATOR II
- Extension: behavioral cartography
 - ▶ Allows to generate the **maximal** set of valuations corresponding to a given **good** behavior
 - ▶ Work in progress!

References I



É. André, T. Chatain, O. De Smet, L. Fribourg, and S. Ruel.

Synthèse de contraintes temporisées pour une architecture d'automatisation en réseau.
In Didier Lime and Olivier H. Roux, editors, *MSR'09*, volume 43 of *Journal Européen des Systèmes Automatisés*, pages 1049–1064. Hermès, 2009.



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

An inverse method for parametric timed automata.
International Journal of Foundations of Computer Science, 20(5):819–836, October 2009.



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

Synthesizing parametric constraints on various case studies using IMITATOR.
Research Report LSV-09-13, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2009.



Étienne André.






IMITATOR: A tool for synthesizing constraints on timing bounds of timed automata.
In Martin Leucker and Carroll Morgan, editors, *ICTAC'09*, volume 5684 of *Lecture Notes in Computer Science*, pages 336–342. Springer, 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, 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.
-  M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang.
Symbolic model checking for probabilistic timed automata.
Information and Computation, 205(7):1027–1077, 2007.
-  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.
-  PRISM web page.
<http://www.prismmodelchecker.org/>.