

# Formal Methods

## Module II: Model Checking

### Ch. 10: **SMT-Based Model Checking**

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it

URL: <http://disi.unitn.it/rseba/DIDATTICA/fm2021/>

Teaching assistant: **Giuseppe Spallitta** – giuseppe.spallitta@unitn.it

M.S. in Computer Science, Mathematics, & Artificial Intelligence Systems  
Academic year 2020-2021

last update: Tuesday 1<sup>st</sup> June, 2021, 11:34

Copyright notice: *some material (text, figures) displayed in these slides is courtesy of R. Alur, M. Benerecetti, A. Cimatti, M. Di Natale, P. Pandya, M. Pistore, M. Roveri, C. Tinelli, and S. Tonetta, who retain its copyright. Some examples displayed in these slides are taken from [Clarke, Grunberg & Peled, "Model Checking", MIT Press], and their copyright is detained by the authors. All the other material is copyrighted by Roberto Sebastiani. Every commercial use of this material is strictly forbidden by the copyright laws without the authorization of the authors. No copy of these slides can be displayed in public without containing this copyright notice.*

- 1 Motivations & Context
- 2 Background
- 3 SMT-Based Bounded Model Checking of Timed Systems
  - Basic Ideas
  - Basic Encoding
  - Improved & Extended Encoding
  - A Case-Study
- 4 Exercises

- 1 Motivations & Context
- 2 Background
- 3 SMT-Based Bounded Model Checking of Timed Systems
  - Basic Ideas
  - Basic Encoding
  - Improved & Extended Encoding
  - A Case-Study
- 4 Exercises

# Motivations

- Model Checking for Timed Systems:
  - relevant improvements and results over the last decades
  - historically, “explicit-state” search style, based on DBMs
    - notable examples: [Kronos](#), [Uppaal](#)
  - More recently, *symbolic* verification techniques:
    - extensions of decision diagrams
    - [CDD](#), [DDD](#), [RED](#), ...
- Key problem: **potential blow up in size**
- A more recent and viable alternative to Binary Decision Diagrams: **SAT-based MC**
  - Bounded Model Checking (BMC), K-induction, IC3/PDR, ...

# Context

## First Idea: SMT-based BMC of Timed Systems

[Audemard et al. 2002], [Sorea, MTCS'02], [Niebert et al., FTRTFT'02]

Leverage the SAT-based BMC approach to Timed Systems by means of **SMT Solvers**

## Extensions

- SMT eventually applied to other SAT-based MC techniques
  - K-Induction
  - interpolant-based
  - IC3/PDR
- SMT applied to a variety of domains:
  - hybrid systems
  - verification of SW (loop invariants/proof obligations, ...)
  - hardware verification
- **Nowadays SMT leading backend technology for FV**

We restrict to BMC for Timed Systems only

# Outline

- 1 Motivations & Context
- 2 **Background**
- 3 SMT-Based Bounded Model Checking of Timed Systems
  - Basic Ideas
  - Basic Encoding
  - Improved & Extended Encoding
  - A Case-Study
- 4 Exercises

# Bounded Model Checking [Biere et al., TACAS'99]

- Given a Kripke Structure  $M$ , an LTL property  $f$  and an integer bound  $k$ , is there an execution path of  $M$  of length (up to)  $k$  satisfying  $f$ ? ( $M \models_k \mathbf{E}f$ )
- Problem converted into the satisfiability of the Boolean formula:

$$[[M]]_k^f := I(s^{(0)}) \wedge \bigwedge_{i=0}^{k-1} R(s^{(i)}, s^{(i+1)}) \wedge (\neg L_k \wedge [[f]]_k^0) \vee \bigvee_{l=0}^k ({}_l L_k \wedge {}_l [[f]]_k^0)$$

$$\text{s.t. } {}_l L_k \stackrel{\text{def}}{=} R(s^{(k)}, s^{(l)}), \quad L_k \stackrel{\text{def}}{=} \bigvee_{l=0}^k {}_l L_k$$

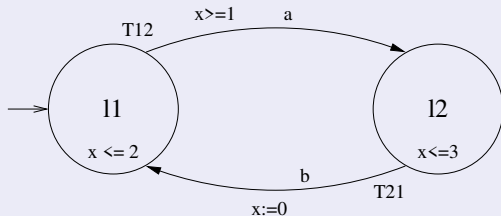
- A satisfying assignment represents a satisfying execution path.
- Test repeated for increasing values of  $k$
- Incomplete
- Very effective for debugging, alternative to OBDDs
- Complemented with **K-Induction** [Sheeran et al. 2000]
- Further developments: **IC3/PDR** [Bradley, VMCAI 2011]

# General Encoding for LTL Formulae

$f$	$[[f]]_k^i$	${}_i[[f]]_k^i$
$p$	$p^{(i)}$	$p^{(i)}$
$\neg p$	$\neg p^{(i)}$	$\neg p^{(i)}$
$h \wedge g$	$[[h]]_k^i \wedge [[g]]_k^i$	${}_i[[h]]_k^i \wedge {}_i[[g]]_k^i$
$h \vee g$	$[[h]]_k^i \vee [[g]]_k^i$	${}_i[[h]]_k^i \vee {}_i[[g]]_k^i$
$Xg$	$[[g]]_k^{i+1}$ if $i < k$ $\perp$ otherwise.	${}_i[[g]]_k^{i+1}$ if $i < k$ ${}_i[[g]]_k^i$ otherwise.
$Gg$	$\perp$	$\bigwedge_{j=\min(i,l)}^k {}_i[[g]]_k^j$
$Fg$	$\bigvee_{j=i}^k [[g]]_k^j$	$\bigvee_{j=\min(i,l)}^k {}_i[[g]]_k^j$
$hUg$	$\bigvee_{j=i}^k \left( [[g]]_k^j \wedge \bigwedge_{n=i}^{j-1} [[h]]_k^n \right)$	$\bigvee_{j=i}^k \left( {}_i[[g]]_k^j \wedge \bigwedge_{n=i}^{j-1} {}_i[[h]]_k^n \right) \vee$ $\bigvee_{j=l}^{i-1} \left( {}_i[[g]]_k^j \wedge \bigwedge_{n=i}^k {}_i[[h]]_k^n \wedge \bigwedge_{n=l}^{j-1} {}_i[[h]]_k^n \right)$
$hRg$	$\bigvee_{j=i}^k \left( [[h]]_k^j \wedge \bigwedge_{n=i}^j [[g]]_k^n \right)$	$\bigwedge_{j=\min(i,l)}^k {}_i[[g]]_k^j \vee$ $\bigvee_{j=i}^k \left( {}_i[[h]]_k^j \wedge \bigwedge_{n=i}^j {}_i[[g]]_k^n \right) \vee$ $\bigvee_{j=l}^{i-1} \left( {}_i[[h]]_k^j \wedge \bigwedge_{n=i}^k {}_i[[g]]_k^n \wedge \bigwedge_{n=l}^j {}_i[[g]]_k^n \right)$



# Timed Automata [Alur and Dill, TCS'94; Alur, CAV'99]



- **Clocks:** real variables (ex.  $x$ )
- **Locations:**
  - **label:** (ex.  $l_1$ ),
  - **invariants:** (conjunctive) constraints on clocks values (ex.  $x \leq 2$ )
- **Switches:**
  - **event labels** (ex.  $a$ ),
  - **clock constraints** (ex.  $x \geq 1$ ),
  - **reset statements** (ex.  $x := 0$ )
- **Time elapse:** all clocks are increased by the same amount

# $\mathcal{LRA}$ -Formulae

[Audemard et al., CADE'02]; [Sorea, MTCS'02]; [Niebert et al., FTRTFT'02]

- $\mathcal{LRA}$ -formulae are Boolean combinations of
  - Boolean variables and
  - linear constraints over real variables (equalities and differences)
    - e.g.,  $(x - 2 \cdot y \geq 4) \wedge ((x = y) \vee \neg A)$
- An interpretation  $\mathcal{I}$  for a  $\mathcal{LRA}$  formula assigns
  - truth values to Boolean variables
  - real values to numerical variables and constants
    - e.g.,  $\mathcal{I}(x) = 3, \mathcal{I}(y) = -1, \mathcal{I}(A) = \perp$
- $\mathcal{I}$  satisfies a  $\mathcal{LRA}$ -formula  $\phi$ , written " $\mathcal{I} \models \phi$ ", iff  $\mathcal{I}(\phi)$  evaluates to true under the standard semantics of Boolean and mathematical operators.
  - E.g.,  $\mathcal{I}((x - 2 \cdot y \geq 4) \wedge ((x = y) \vee \neg A)) = \top$

- **Bottom level:** a  $\mathcal{T}$ -Solver for sets of  $\mathcal{LRA}$  constraints
  - E.g.  $\{\dots, z_1 - x_1 \leq 6, z_2 - x_2 \geq 8, x_1 = x_2, z_1 = z_2, \dots\} \implies \text{unsat.}$
  - Combination of symbolic and numerical algorithms (equivalence class building, Belman-Ford, Simplex)
- **Top level:** a CDCL procedure for propositional satisfiability
  - mathematical predicates treated as propositional atoms
  - invokes  $\mathcal{T}$ -Solver on every assignment found
  - used as an enumerator of assignments
  - lots of enhancements

(see chapter on SMT)

- 1 Motivations & Context
- 2 Background
- 3 SMT-Based Bounded Model Checking of Timed Systems**
  - Basic Ideas
  - Basic Encoding
  - Improved & Extended Encoding
  - A Case-Study
- 4 Exercises

- 1 Motivations & Context
- 2 Background
- 3 SMT-Based Bounded Model Checking of Timed Systems**
  - **Basic Ideas**
  - Basic Encoding
  - Improved & Extended Encoding
  - A Case-Study
- 4 Exercises

# SMT-Based BMC for Timed Systems

Independently developed approaches (2002):

- [Audemard et al. FORTE'02]: encoding into  $\mathcal{LRA}$ 
  - all LTL properties
- [Sorea, MTCS'02]: encoding into  $\mathcal{LRA}$ 
  - based on automata-theoretic approach for LTL
- [Niebert et al., FTRTFT'02]: encoding into  $\mathcal{DL}$ 
  - limited to reachability

## Disclaimer

These slides are adapted from [Audemard et al. FORTE'02]:

G. Audemard, A. Cimatti, A. Kornilowicz, R. Sebastiani  
*Bounded Model Checking for Timed Systems*,  
proc. FORTE 2002, Springer

freely available as <http://eprints.biblio.unitn.it/124/>

# BMC for Timed Systems

## Basic ingredients:

- An extension of propositional logic expressive enough to represent timed information: “ $\mathcal{LRA}$ -formulae”
- A  $\text{SMT}(\mathcal{LRA})$  solver for deciding  $\mathcal{LRA}$ -formulae  
⇒ e.g., the **MATHSAT** solver
- An **encoding** from timed BMC problems into  $\mathcal{LRA}$ -formulae
  - $\mathcal{LRA}$ -satisfiable iff an execution path within the bound exists

- 1 Motivations & Context
- 2 Background
- 3 **SMT-Based Bounded Model Checking of Timed Systems**
  - Basic Ideas
  - **Basic Encoding**
  - Improved & Extended Encoding
  - A Case-Study
- 4 Exercises



# The encoding

Given a **timed automaton**  $A$  and a **LTL formula**  $f$ :

- The encoding  $[[A, f]]_k$  is obtained following the same schema as in propositional BMC:

$$[[A, f]]_k := I(s^{(0)}) \wedge \bigwedge_{i=0}^{k-1} R(s^{(i)}, s^{(i+1)}) \wedge (\neg L_k \wedge [[f]]_k^0) \vee \bigvee_{l=0}^k ({}_l L_k \wedge {}_l [[f]]_k^0)$$

- $[[M, f]]_k$  is a  $\mathcal{LRA}$ -formula, where
  - Boolean variables encode the **discrete part** of the state of the automaton
  - constraints on real variables represent the **temporal part** of the state

# Encoding: Boolean Variables

- **Locations:** an array  $\underline{l}$  of  $n \stackrel{\text{def}}{=} \lceil \log_2(|L|) \rceil$  Boolean variables
  - $\underline{l}_i$  holds iff the system is in the location  $l_i$
  - ex: “ $\neg \underline{l}_i[3] \wedge \underline{l}_i[2] \wedge \neg \underline{l}_i[1] \wedge \underline{l}_i[0]$ ” means “the system is in location  $\underline{l}_3$ ”
  - “ $(\underline{l}_i = \underline{l}_j)$ ” stands for “ $\bigwedge_n (\underline{l}_i[n] \leftrightarrow \underline{l}_j[n])$ ”,
  - “primed” variables  $\underline{l}_i'$  to represent location after transition
- **Events:** for each event  $a \in \Sigma$ , a Boolean variable  $\underline{a}$ 
  - $\underline{a}$  holds iff the system executes a switch with event  $a$ .
- **Switches:** for each switch  $\langle l_i, a, \varphi, \lambda, l_j \rangle \in E$ , a Boolean variable  $T$ ,
  - $T$  holds iff the system executes the corresponding switch
- **Time elapse and null transitions:** two variables  $T_\delta$  and  $T_{null}^j$ 
  - $T_\delta$  holds iff time elapses by some  $\delta > 0$
  - $T_{null}^j$  holds if and only if  $A_j$  does nothing (specific for automaton  $A_j$ )

Note: also for events, switches&transitions it is possible to use arrays of Boolean variables of size  $\lceil \log_2(|\Sigma|) \rceil$ ,  $\lceil \log_2(|E| + 2) \rceil$  respectively

# Encoding: Clock Values and Constraints

- **Clocks values**  $x$  are “normalized” wrt absolute time  $(x - z)$ :
  - a clock value is written as difference  $x - z$
  - $z$  represents (the negation of) **the absolute time**
  - “offset” variable  $x$  represents (the negation of) **the absolute time when the clock was reset**
- **Clock constraints** reduce to  $(x - z \bowtie c)$ ,  $\bowtie \in \{\leq, \geq, <, >\}$ ,  $c \in \mathbb{Z}$
- **Clock reset conditions** reduce to  $(x := z)$
- **Clock equalities** like  $(x_k = x_l)$  reduce to  $(x_k - z_k = x_l - z_l)$ 
  - appear only in loops
  - only place where full  $\mathcal{LRA}$  is needed (rather than  $\mathcal{DL}$ ) $\implies$  for invariant checking (no loops)  $\mathcal{DL}$  suffices
- Encoding the **effect of transitions**:
  - with a time elapse transition
    - $z' < z$ , and  $x' = x$
  - otherwise:
    - $z' = z$ , absolute time does not elapse
    - $x' = z'$ , if the clock is reset
    - $x' = x$ , if the clock is not reset

# Encoding: Initial Conditions

Initial condition  $I(s)$ :

- Initially, the automaton is in an initial location:

$$\bigvee_{l_i \in L^0} \underline{l_i}$$

- Initially, clocks have a null value:

$$\bigwedge_{x \in X} (x = z)$$

# Encoding: Invariants

## Transition relation $R(s, s')$ : Invariants

- Always, being in a location implies the corresponding constraints:

$$\bigwedge_{l_j \in L} (l_j \rightarrow \bigwedge_{\psi \in I(l_j)} \psi),$$

# Encoding: Transitions

Transition relation  $T(s, s')$ :

- Switches:

$$T \stackrel{\text{def}}{=} \bigwedge_{\langle l_i, a, \varphi, \lambda, l_j \rangle \in E} \left( \underline{l}_i \wedge \underline{a} \wedge \varphi \wedge \underline{l}_j' \wedge \bigwedge_{x \in \lambda} (x' = z') \wedge \bigwedge_{x \notin \lambda} (x' = x) \wedge (z' = z) \right)$$

- Time elapse:

$$T_\delta \rightarrow \left( (z' - z < 0) \wedge (\underline{l}' = \underline{l}) \wedge \bigwedge_{x \in X} (x' = x) \wedge \bigwedge_{a \in \Sigma} \neg \underline{a} \right)$$

- Null transition:

$$T_{null}^j \rightarrow \left( (z' = z) \wedge (\underline{l}' = \underline{l}) \wedge \bigwedge_{x \in X} (x' = x) \wedge \bigwedge_{a \in \Sigma} \neg \underline{a} \right)$$

## Encoding: Relations between Transitions

- Mutual exclusion between events:

$$\bigwedge_{a_k, a_r \in \Sigma, a_k \neq a_r} (\neg \underline{a}_k \vee \neg \underline{a}_r)$$

- At least one transition takes place:

$$T_{null}^j \vee T_\delta \vee \bigvee_{T \in E} T$$

- Mutual exclusion between transitions:

$$\bigwedge_{T_k, T_r \in EU\{T_{null}^j\} \cup \{T_\delta\}, T_k \neq T_r} (\neg T_k \vee \neg T_r)$$

If events and transitions are encoded via arrays of Booleans, mutual exclusion constraints are not needed

# Automata Product Construction

- The encoding is compositional wrt. product of automata
- The encoding of  $A = A_1 || A_2$  is given by the **conjunction of the encodings of  $A_1$  and  $A_2$** , plus a few extra axioms
- Mutual exclusion between events that are local

$$\bigwedge_{\substack{a_1 \in \Sigma_1 \setminus \Sigma_2 \\ a_2 \in \Sigma_2 \setminus \Sigma_1}} (\neg a_1 \vee \neg a_2)$$

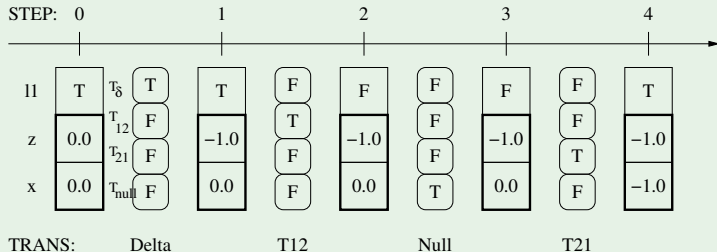
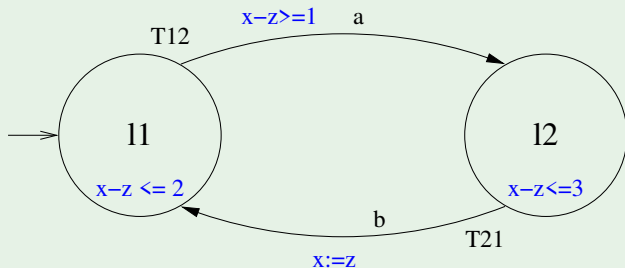
- Forcing system activity:

$$\bigvee_{j=0}^{N-1} \neg T_{null}^j$$

- one distinct  $T_{null}^j$  for each automaton  $A_j$
- $T_{\delta}$  is common to all automata  $A_j$



# A Simple Example



# Outline

- 1 Motivations & Context
- 2 Background
- 3 SMT-Based Bounded Model Checking of Timed Systems**
  - Basic Ideas
  - Basic Encoding
  - Improved & Extended Encoding**
  - A Case-Study
- 4 Exercises

## Adding Global Variables

Dealing with some global variable  $v$  on discrete domain:

- A switch  $T \stackrel{\text{def}}{=} \langle l_i, a, \varphi, \lambda, l_j \rangle$  can
  - be subject to a condition  $\psi(v)$ 
    - $\Rightarrow$  add  $T \rightarrow \psi(v)$
    - assign  $v$  to some value  $n$  or keep its value
      - $\Rightarrow$  add  $T \rightarrow (v' = n)$  or add  $T \rightarrow (v' = v)$
  - $T_\delta$  maintains the value of  $v$ :
    - $\Rightarrow$  add  $T_\delta \rightarrow (v' = v)$
  - $T_{null}^j$  imposes no constraint on  $v$ :
    - $\Rightarrow$  add nothing (for  $A_j$ )

## Customization of MATHSAT

- Limit Boolean variable-selection heuristic to pick **transition variables**, in forward order

# Encoding: Optimizations

## Boolean Propagation of Math Constraints:

Idea: add small and mathematically-obvious lemmas

$$\neg(z' = z) \leftrightarrow (z' - z < 0)$$

$$\bigwedge_{x \in X} (\neg(x = z) \leftrightarrow (x - z > 0))$$

$$\bigwedge_{x \in X} \neg(x' = x) \leftrightarrow (x' - x < 0)$$

$\bigwedge_{x \in X} (x = z)$	$\wedge$	$(x' = x)$	$\wedge$	$(z' = z)$	$\rightarrow$	$(x' = z')$
$\bigwedge_{x \in X} (\neg(x = z))$	$\wedge$	$(x' = x)$	$\wedge$	$(z' = z)$	$\rightarrow$	$\neg(x' = z')$
$\bigwedge_{x \in X} (x = z)$	$\wedge$	$\neg(x' = x)$	$\wedge$	$(z' = z)$	$\rightarrow$	$\neg(x' = z')$
$\bigwedge_{x \in X} (x = z)$	$\wedge$	$(x' = x)$	$\wedge$	$\neg(z' = z)$	$\rightarrow$	$\neg(x' = z')$
$\bigwedge_{x \in X} (x' = x)$	$\wedge$	$(z' - z < 0)$	$\wedge$	$(x - z > 0)$	$\rightarrow$	$(x' - z' > 0)$
$\bigwedge_{x \in X} (z' = z)$	$\wedge$	$\neg(x - z > 0)$	$\wedge$	$(x' - x < 0)$	$\rightarrow$	$\neg(x' - z' > 0)$
$\bigwedge_{x \in X} (x - z \boxtimes c)$	$\wedge$	$(x' = x)$	$\wedge$	$(z' = z)$	$\rightarrow$	$(x' - z' \boxtimes c)$
$\bigwedge_{x \in X} (\neg(x - z \boxtimes c))$	$\wedge$	$(x' = x)$	$\wedge$	$(z' = z)$	$\rightarrow$	$\neg(x' - z' \boxtimes c)$

$\implies$  force assignments by unit-propagation,

$\implies$  saves calls to the  $\mathcal{T}$ -Solvers

# Encoding Variants

## Shortening counter-examples:

- **Collapsing consequent time elapsing transitions:**

- $s \xrightarrow{\delta} s, s \xrightarrow{\delta'} s$  reduced to  $s \xrightarrow{\delta+\delta'} s$

- add  $\neg T_\delta \vee \neg T'_\delta$  to transition relation  $R(s, s')$

⇒ implements the notion of “non-Zeno-ness” (see previous chapter)

- **Allow multiple parallel transitions**

- remove mutex between labels local to processes

⇒ allows a form of parallel progression

Remark: may change the notion of “next step”

# Encoding Variants (cont.)

## A limited form of symmetry reduction

If  $N$  automata are symmetric (frequent with protocol verification):

- Intuition: restrict executions s.t.
  - At step 0 only  $A_0$  can move
  - At step 1 only  $A_0, A_1$  can move
  - At step 2 only  $A_0, A_1, A_2$  can move
  - ...

- for step  $i < N - 1$ ,

we drop the disjunct  $\neg T_{null}^{i+1}(i) \vee \dots \vee \neg T_{null}^{N-1}(i)$ :

$$\text{set } \bigvee_{j=0}^{\min(i, N-1)} \neg T_{null}^j(i) \text{ rather than } \bigvee_{j=0}^{N-1} \neg T_{null}^j(i)$$

⇒ drops “symmetric” executions

⇒ reduces the search space of a up to  $2^{N(N-1)/2}$  factor!

# Outline

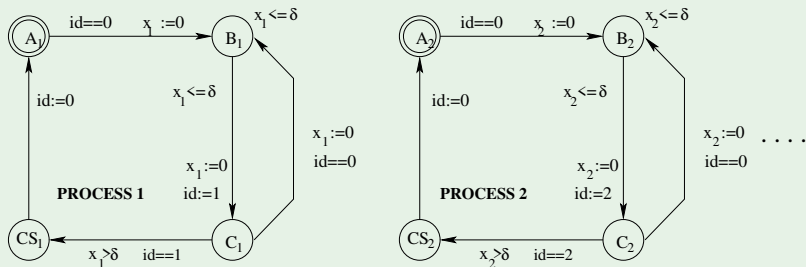
- 1 Motivations & Context
- 2 Background
- 3 SMT-Based Bounded Model Checking of Timed Systems**
  - Basic Ideas
  - Basic Encoding
  - Improved & Extended Encoding
  - A Case-Study**
- 4 Exercises



# A Case-study: Fischer's Protocol

## A Mutual-Exclusion Real-Time Protocol

- N identical processes accessing one critical section
- shared variable  $id \in \{0, 1, 2, \dots, N\}$ : process identifier (0: none)
  - when entering wait state  $C_j$ , agent  $A_j$  writes its code on  $id$
  - if  $id = j$  after  $\delta$ , then  $A_j$  can enter the critical session
- Two properties under test
  - Reachability:  $EF \bigwedge_i P_i.C$  (reached in  $N+1$  steps)
  - Fairness:  $E \neg (GFP_i.B \rightarrow GFP_i.CS)$  (reached in  $N+5$  steps)



## Fischer's protocol: (cont.)

### Exercise:

- Why is  $\mathbf{EF} \bigwedge_i P_i.C$  reached in  $N+1$  steps?
- Why is  $\mathbf{E}\neg(\mathbf{GFP}_i.B \rightarrow \mathbf{GFP}_i.CS)$  reached in  $N+5$  steps?

(See [Audemard et al, FORTE'02] for the solution.)

# Fischer's protocol: (reachability)

$$M \models_k \mathbf{EF} \bigwedge_i P_i.C$$

N	MATHSAT		MATHSAT,Sym		DDD		UPPAL		KRONOS		RED		RED,Sym	
	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size
3	0.05	2.9	0.04	2.9	0.11	106	0.01	1.7	0.01	0.8	0.23	2.0	0.19	2.0
4	0.09	3.0	0.08	3.0	0.14	106	0.02	1.9	0.02	2.2	1.00	2.1	0.70	2.1
5	0.20	3.2	0.16	3.2	0.24	106	0.21	1.9	0.09	19	3.70	2.2	2.00	2.4
6	0.60	3.7	0.23	3.7	0.47	106	3.44	6.7	0.39	236	12.00	2.7	5.20	3.1
7	3.20	4.2	0.36	4.2	1.30	106	153	54		MEM	38	4.0	12	4.7
8	29	4.9	0.52	4.9	3.96	106	TIME				121	7.6	26	7.8
9	343	5.9	0.75	5.9	14	106					416	16.6	49	13.3
10	3331	6.5	1.01	6.5	62	106					1382	39	90	23
11	TIME		1.39	7.0	691	106					TIME		157	38
12			1.89	7.5		MEM							266	63
13			2.44	8.2									439	100
14			3.24	8.9									709	155
15			4.11	9.7									1118	225
16			5.10	10.7									1717	342
17			6.30	11.7									2582	492
18			8.00	12.9									TIME	
19			9.50	14.2										

(MATHSAT times are sum of all instances up to  $k$ )

# Fischer's protocol (liveness violation)

$$M \models_k \mathbf{E} \neg (\mathbf{GFP}_i . B \rightarrow \mathbf{GFP}_i . CS)$$

$k \setminus N$	MATHSAT					MATHSAT with Boehm heuristic				
	2	3	4	5	6	2	3	4	5	6
2	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.02
3	0.01	0.02	0.01	0.01	0.03	0.01	0.01	0.02	0.03	0.04
4	0.01	0.02	0.02	0.02	0.04	0.01	0.02	0.04	0.07	0.17
5	0.02	0.03	0.05	0.09	0.18	0.01	0.03	0.09	0.30	1.16
6	0.03	0.10	0.21	0.54	1.35	0.02	0.07	0.31	1.52	7.74
7	0.04	0.26	0.97	3.20	9.83	0.02	0.18	1.19	7.14	45.00
8		0.65	4.80	19.72	70.70		0.06	4.70	33.50	242.00
9			5.55	112.17	478.00			0.61	165.90	1348.00
10				303.17	3086.00				9.92	7824.00
11					5002.00					252.00
$\Sigma$	0.12	1.08	11.62	438.93	8648.15	0.07	0.37	6.98	218.40	9720.13

# Outline

- 1 Motivations & Context
- 2 Background
- 3 SMT-Based Bounded Model Checking of Timed Systems
  - Basic Ideas
  - Basic Encoding
  - Improved & Extended Encoding
  - A Case-Study
- 4 Exercises**

# Proposed Exercise

## Proposed Exercise

- Consider the Train-gate-controller example from [Alur CAV'99] (see previous chapter)
  - Encode the Initial state formula
  - Encode the transition relation
  - Encode the BMC problem for the formula  $\mathbf{G}(s_2 \rightarrow t_2)$
- As above, reducing the delay time for the controller from 1 to 0.5
  - what happens?
  - in how many steps?
- Encode the above into MathSAT