Synthesis and Analysis of Timed Communicating Systems

S. Akshay\textsuperscript{1,2}, Benedikt Bollig\textsuperscript{1}, Paul Gastin\textsuperscript{1}

\textsuperscript{1}LSV, ENS Cachan, CNRS, France
\textsuperscript{2}Institute of Mathematical Sciences, Chennai, India

Graduiertenkolleg AlgoSyn, Aachen
December 5, 2007
Formal methods

specification
synthesis
modeling & verification
code generation

/* Sending Process */
initsend();
putint(2);
send("receiver",4,99);

/* Receiving Process */
int num;
recv(99);
getint(&num);

G (1!2 → F 2!1)
The automata/systems scene

Sequential systems

Words: $a_1a_2a_3\ldots a_n \in \Sigma^*$

Timed systems

Timed words: $(a_1, t_1)(a_2, t_2)\ldots$

Distributed systems

Timed and distributed systems??

Distributed systems

Partial orders:

$\xrightarrow{a_1} \xrightarrow{a_2} \xrightarrow{a_3} \xrightarrow{a_4}$

Timed and distributed systems??

Timed partial orders??
In the beginning there were ...

\[ abaabbb \in \{a, b\}^* \]

\[ \exists X \ a(x_{\text{min}}) \wedge x_{\text{min}} \in X \wedge x_{\text{max}} \in X \wedge \forall x (x \in X \leftrightarrow \neg (x + 1 \in X)) \]

[Büchi, Elgot 1960]
Timed setting

\[(a, 0.2) (b, 1) (a, 1.5) (a, 1.6) (b, 2.9) (b, 3) (b, 5) \in (\{a, b\} \times \mathbb{R}_{\geq 0}^*)\]

\[\exists X \ a(x_{min})\]
\[\land x_{min} \in X\]
\[\land x_{max} \in X\]
\[\land \forall x (x \in X \leftrightarrow \neg(x + 1 \in X))\]
\[\land \forall x (\neg x \in X \rightarrow \delta(x, \text{Next}_b(x)) \in [1, 2])\]

[D’Souza 2003]
Distributed setting

∀x(1!2(req)(x) → ∃y(x ≤ y ∧ 2!1(ack)(y)))

[Genest & Kuske & Muscholl 2004]
Goal in this work

- To introduce timing in MSCs as a formalism to describe machines with timing and concurrence.
- Obtain a similar equivalence between automata implementation and logical specification.
Message Sequence Charts

- Attractive *visual* formalism to describe interaction between processes.
- Particularly useful in early system design.
- Example:

![Message Sequence Chart Diagram]

Processes: User1, User2, Spool, Printer
Events: $e$, $f$, $g$, $u$, ...
Actions: $\text{User1}!\text{Spool}(\text{print})$, $\text{Printer}?\text{Spool}(\text{print})$, ...
Messages: $(e,f)$, $(g,h)$, $(u,v)$, ...

... and a bit more formally:

Labelled partial order $(E, \leq, \ell)$ with
$E = \{e, f, g, u, \ldots\}$
$\leq = \{(e,f), (f,u), (e,v), \ldots\}$
$\ell : E \rightarrow \{\text{User1}!\text{Spool}(\text{print}), \text{Printer}?\text{Spool}(\text{print}), \ldots\}$
Message Sequence Charts

- Attractive *visual* formalism to describe interaction between processes.
- Particularly useful in early system design.
- Example:

Processes: User1, User2, Spool, Printer
Events: $e$, $f$, $g$, $u$, ...
Actions: User1!Spool(print), Printer?Spool(print), ...
Messages: $(e,f)$, $(g,h)$, $(u,v)$, ...

... and a bit more formally:

Labelled partial order $(E, \leq, \ell)$ with
\[
E = \{ e, f, g, u, \ldots \} \\
\leq = \{ (e,f), (f,u), (e,v), \ldots \} \\
\ell : E \to \{ \text{User1!Spool(print)}, \text{Printer?Spool(print)}, \ldots \}
Introducing timing (T-MSC)
Attach time stamps (which are non-negative real numbers) to events.

This is the natural formalism for timing which extends from timed words.

However, it is not natural for specification by engineers! So we introduce another model ...
MSC with timing constraints (TC-MSC)

- We attach time intervals to “selected pairs” of events.
- We can restrict the pairs and thus control timing. This also will depend on and determine our logic and automaton models.
- This is natural for specification.
Allowed timing intervals
Allowed timing intervals

$(3,6)$

Allows timing between all pairs of events that form messages.
Allowed timing intervals

Allows timing between an event and another whose label is the previous appearance of that action label from this event.
Allowed timing intervals

Allows timing between an event and another whose label is the previous appearance of that action label from this event.
Allowed timing intervals

Allows timing between an event and another whose label is the very next appearance of that action label.
Allowed timing intervals

Allows timing between an event and another whose label is the very next appearance of that action label.
TC-MSC, T-MSC, and timed words
TC-MSC, T-MSC, and timed words

Akshay, Bollig, Gastin (LSV & IMS)

Timed Communicating Systems

Graduiertenkolleg AlgoSyn
TC-MSC, T-MSC, and timed words

\[ (0,2) \]
\[ [1,2] \]
\[ [1,4] \]
\[ [2,3] \]
TC-MSC, T-MSC, and timed words

\( (p!q, 1)(q!p, 1.2)(p!q, 3)(q!p, 3)(q?p, 4)(p?q, 5) \)

\( (p!q, 1)(q!p, 1.2)(q!p, 3)(p!q, 3)(q?p, 4)(p?q, 5) \)

\( (p!q, 1)(q!p, 1.2)(p!q, 3)(q!p, 3)(p?q, 5)(q?p, 5) \)

\( (p!q, 1)(q!p, 1.2)(q!p, 3)(p!q, 3)(q?p, 5)(p?q, 5) \)
Event-clock communicating finite-state machine

User

\[ S_1 \]

!pswd

\[ S_2 \]

?wrong

\[ Prev_{pswd} \in [5, 7] \]

?correct

\[ S_3 \]

Server

\[ t_1 \]

!correct

?pswd

\[ Msg \in [0, 4] \]

!wrong
Event-clock communicating finite-state machine

User

\[ S_1 \]

$pwd$ \[ ! \]

\[ S_2 \]

Wrong $pwd \in [5, 7]$ \[ ? \]

$correct$ \[ ! \]

\[ S_3 \]

Server

\[ t_1 \]

$pwd$ \[ ? \]

Wrong \[ ! \]

\[ t_2 \]

\[ t_1 \]

Wrong $pwd \in [0, 4]$ \[ ! \]

User Server
Event-clock communicating finite-state machine

User

\(s_1\)

\(s_2\)

\(s_3\)

?correct

?wrong

\(Prev_{pswd} \in [5, 7]\)

Server

\(t_1\)

\(t_2\)

?correct

?wrong

!correct

!pswd

!wrong

\(Msg \in [0, 4]\)

Prev

Akshay, Bollig, Gastin (LSV & IMS)  Timed Communicating Systems  Graduiertenkolleg AlgoSyn  13 / 28
Event-clock communicating finite-state machine

User

\[ s_1 \]

\[ s_2 \]

\[ s_3 \]

\(?\text{wrong} \overset{!\text{pswd}}{\rightarrow} [5, 7]\)

\(?\text{correct} \]

Server

\[ t_1 \]

\[ t_2 \]

\(?\text{pswd} \overset{!\text{correct}}{\rightarrow} [0, 4]\)

\(?\text{wrong} \]

Prev \(?\text{pswd} \in [5, 7]\)

Msg \(?\text{pswd} \in [0, 4]\)

!correct

!wrong

User Server

1 \rightarrow 3

pswd
Event-clock communicating finite-state machine

User

- $s_1 \rightarrow s_2$ with $?\text{pswd} \in [5, 7]
- $s_2 \rightarrow s_3$ with $?\text{correct}$

Server

- $t_1 \rightarrow t_2$ with $!\text{correct}$
- $t_2 \rightarrow s_2$ with $?\text{pswd Msg} \in [0, 4]$ and $!\text{wrong}$

Prev $!\text{pswd}$

$p$ 

$t_1$

$t_2$

$s_3$

Akshay, Bollig, Gastin (LSV & IMS) 
Timed Communicating Systems 
Graduiertenkolleg AlgoSyn
Event-clock communicating finite-state machine

User

\[ \text{State } s_1 \]

\[ \text{Input: } !\text{pswd} \]

\[ \text{Output: } \text{Prev}_!\text{pswd} \in [5, 7] \]

\[ \text{State } s_2 \]

\[ \text{Input: } ?\text{correct} \]

\[ \text{State } s_3 \]

Server

\[ \text{State } t_1 \]

\[ \text{Input: } !\text{correct} \]

\[ \text{Output: } !\text{wrong} \]

\[ \text{State } t_2 \]

\[ \text{Input: } ?\text{pswd} \]

\[ \text{Output: } \text{Msg} \in [0, 4] \]

Akshay, Bollig, Gastin (LSV & IMS)
Timed Communicating Systems
Graduiertenkolleg AlgoSyn 13 / 28
Event-clock communicating finite-state machine

Akshay, Bollig, Gastin (LSV & IMS)  
Timed Communicating Systems  
Graduiertenkolleg AlgoSyn  
13 / 28
Event-clock communicating finite-state machine

Akshay, Bollig, Gastin (LSV & IMS)  Timed Communicating Systems  Graduiertenkolleg AlgoSyn
Event-clock communicating finite-state machine

User

\[ S_1 \]
\[ !\text{pswd} \]
\[ \text{?wrong} \]
\[ \text{Prev}_{\text{pswd}} \in [5, 7] \]

\[ S_2 \]

\[ \text{?correct} \]

\[ S_3 \]

Server

\[ t_1 \]
\[ !\text{correct} \]
\[ \text{!wrong} \]
\[ \text{?pswd} \]
\[ \text{Msg} \in [0, 4] \]

\[ t_2 \]

User

1

7

8

9

10

Server

3

5

pswd

wrong

correct
Event-clock communicating finite-state machine

User

- $s_1$:
  - $!\text{pswd}$
  - $?\text{correct}$

- $s_2$:
  - $?\text{wrong}$

- $s_3$:
  - $?\text{correct}$

$\text{Prev}_{\text{pswd}} \in [5, 7]$ and $\text{Msg} \in [0, 4]$

Server

- $t_1$:
  - $!\text{correct}$
  - $!\text{wrong}$

User Server

1. User sends $\text{pswd}$ to Server
2. Server checks $\text{pswd}$ against $\text{Prev}_{\text{pswd}}$ and $\text{Msg}$
3. If correct, Server sends $\text{ correct}$ to User
4. If wrong, Server sends $\text{ wrong}$ to User
Event-clock communicating finite-state machine

User

\[ S_1 \]

!\text{pswd}

?\text{wrong} \quad \text{Prev}_{\text{pswd}} \in [5, 7]

\[ S_2 \]

?\text{correct}

\[ S_3 \]

Server

\[ t_1 \]

!\text{correct}

?\text{pswd} \quad \text{Msg} \in [0, 4]

\[ t_2 \]

!\text{wrong}

Akshay, Bollig, Gastin (LSV & IMS)
Goal in this work

Timed MSCs

TMSO over Timed MSCs

Synthesis

EC-CFMs over Timed-MSCs
Definition

Monadic second-order logic (MSO):

\[ \varphi ::= x \leq y \mid x \lesssim_{\text{proc}} y \mid x \lesssim_{\text{msg}} y \]

\[ \text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \]

Akshay, Bollig, Gastin (LSV & IMS)  Timed Communicating Systems
Graduiertenkolleg AlgoSyn 15 / 28
MSO logic vs. automata – The untimed case

Theorem

Monadic second-order logic (MSO):

\[ \varphi ::= x \leq y \mid x \leq_{\text{proc}} y \mid x <_{\text{msg}} y \]

\[ \text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \]

- any implementation has a specification for any CFM \( A \), there is an MSO formula \( \varphi \) such that \( A \) recognizes precisely the models of \( \varphi \)
- not every specification is implementable \[\text{[B. \& Leucker 2004]}\]
MSO logic vs. automata – The untimed case

Theorem

Monadic second-order logic (MSO):

\[ \varphi ::= x \leq y \mid x \preceq_{\text{proc}} y \mid x \preceq_{\text{msg}} y \]

\[ \text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \]

- **not every implementation has a specification**
- **not every specification is implementable**
Theorem

Monadic second-order logic (MSO):

\[ \varphi ::= x \leq y \mid x \leq_{\text{proc}} y \mid x <_{\text{msg}} y \]

\[ \text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \]

If we restrict to \( \exists \)-bounded sets [Genest & Kuske & Muscholl 2004]:

- every implementation has a specification
- every specification is implementable
MSO logic vs. automata – The untimed case

**Theorem**

Existential monadic second-order logic (EMSO):

\[ \varphi ::= x \leq y \mid x \preceq_{\text{proc}} y \mid x \prec_{\text{msg}} y \]

\[ \text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \]

- every implementation has a specification
- every specification is implementable  [B. & Leucker 2004]
MSO logic vs. automata – The timed case

**Definition**

**Timed monadic second-order logic (TMSO):**

\[ \varphi ::= x \leq y \mid x <_\text{proc} y \mid x <_\text{msg} y \]

\[ \delta(x, \text{Prev}_{\text{act}}(x)) \in I \mid \delta(x, \text{Next}_{\text{act}}(x)) \in I \mid \delta(x, \text{Msg}(x)) \in I \]

\[ \text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \]

\[ \forall x(p!q(x) \rightarrow \delta(x, \text{Msg}(x)) \in [0,5]) \]

\[ \land \exists x(p!q(x) \land \delta(x, \text{Next}_{\text{q} \rightarrow \text{p}}(x)) \in [1,3]) \]
MSO logic vs. automata – The **timed** case

**Theorem**

**Timed** monadic second-order logic ($\text{TMSO}$):

\[
\varphi ::= x \leq y \mid x \prec_{\text{proc}} y \mid x \prec_{\text{msg}} y
\]

\[
\delta(x, \text{Prev}_{\text{act}}(x)) \in I \mid \delta(x, \text{Next}_{\text{act}}(x)) \in I \mid \delta(x, \text{Msg}(x)) \in I
\]

\[
\text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi
\]

If we restrict to $\exists$-bounded channels:

- every implementation has a specification
- every specification is implementable
Theorem

Timed monadic second-order logic (TMSO):

\[ \varphi ::= x \leq y \mid x \prec_{\text{proc}} y \mid x \prec_{\text{msg}} y \]

\[ \delta(x, \text{Prev}_{\text{act}}(x)) \in I \mid \delta(x, \text{Next}_{\text{act}}(x)) \in I \mid \delta(x, \text{Msg}(x)) \in I \]

\[ \text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \]

More precisely ...

For a set \( L \) of \textit{untimed-}\( \exists \)-B-bounded T-MSCs, the following are equivalent:

- There is an EC-CFM recognizing \( L \).
- There is a TMSO formula \( \varphi \) defining \( L \).
MSO logic vs. automata – The timed case

Theorem

Timed monadic second-order logic (TMSO):

\[ \varphi ::= x \leq y \mid x \prec_{\text{proc}} y \mid x \prec_{\text{msg}} y \]

\[ \delta(x, \text{Prev}_a(x)) \in I \mid \delta(x, \text{Next}_a(x)) \in I \mid \delta(x, \text{Msg}(x)) \in I \]

\[ \tau(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \forall x \varphi \mid \exists X \varphi \]

This is, however, not completely intuitive:

\[ \forall x (p!q(x) \lor q?p(x)) \land \forall x (p!q(x) \rightarrow \delta(x, \text{Next}_p(x)) \in [0, 0) \land \delta(x, \text{Msg}(x)) \in (0, \infty)) \]

defines an untimed-∃-1-bounded set of T-MSCs.
MSO logic vs. automata – The timed case

Theorem

Existential timed monadic second-order logic ($\text{ETMSO}$):

\[ \varphi ::= x \leq y \mid x \preceq_{\text{proc}} y \mid x \prec_{\text{msg}} y \]
\[ \delta(x, \text{Prev}_{\text{act}}(x)) \in I \mid \delta(x, \text{Next}_{\text{act}}(x)) \in I \mid \delta(x, \text{Msg}(x)) \in I \]
\[ \text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi \]

- every implementation has a specification
- every specification is implementable
MSO logic vs. automata – The **timed** case

**Theorem**

**Existential timed monadic second-order logic (ETMSO):**

\[
\varphi ::= \quad x \leq y \mid x \prec_{\text{proc}} y \mid x \prec_{\text{msg}} y
\]

\[
\delta(x, \text{Prev}_{\text{act}}(x)) \in I \mid \delta(x, \text{Next}_{\text{act}}(x)) \in I \mid \delta(x, \text{Msg}(x)) \in I
\]

\[
\text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists x \varphi \mid \exists X \varphi
\]

- every implementation has a specification
- every specification is implementable
- but how about effectiveness?
Sketch of proof

TMSO over Timed MSCs to show EC-CFMs over Timed MSCs
Sketch of proof

TMSO over Timed MSCs

EC-CFMs over Timed MSCs

MSO over MSCs

CFMs over MSCs
Sketch of proof

TMSO over Timed MSCs

\[ T \models \varphi \text{ over alphabet } \text{Act} \]

EC-CFMs over Timed MSCs

\[ T^{\text{ext}} \models \varphi^{\text{ext}} \text{ over } \text{Act} \times \text{TC} \]

MSO over MSCs

CFMs over MSCs
Sketch of proof

\( T \models \varphi \) over alphabet \( \text{Act} \)

\( \delta(x, \text{Msg}(x)) \in I \)

\( T_{\text{ext}} \models \varphi_{\text{ext}} \) over \( \text{Act} \times \text{TC} \)

\( \forall (act, tc) \in \text{P} \)

\( tc \models (\text{Msg}, I) \)

MSO over MSCs

EC-CFMs over Timed MSCs

Akshay, Bollig, Gastin (LSV & IMS)

Timed Communicating Systems

Graduiertenkolleg AlgoSyn
Sketch of proof

TMSO over Timed MSCs

\[ T \models \varphi \]

\[ T^{\text{ext}} \models \varphi^{\text{ext}} \]

EC-CFMs over Timed MSCs

\[ T \text{ is acc. by } A \]

\[ T^{\text{ext}} \text{ is acc. by } A^{\text{ext}} \]
Sketch of proof

$T \models \varphi$

$T^{\text{ext}} \models \varphi^{\text{ext}}$

$T$ is acc. by $A$

$T^{\text{ext}}$ is acc. by $A^{\text{ext}}$

TMSO over Timed MSCs $\rightarrow$ EC-CFM over Timed MSCs

MSO over MSCs $\rightarrow$ CFMs over MSCs
Goal in this work

- Timed MSCs
- Synthesis
- TMSO over Timed MSCs
- EC-CFMs over Timed-MSCs
Goal in this work

- Timed MSCs
- TMSO over Timed MSCs
- EC-CFMs over Timed-MSCs

Synthesis

Analysis

Akshay, Bollig, Gastin (LSV & IMS)

Timed Communicating Systems

Graduiertenkolleg AlgoSyn
Emptiness for CFMs is undecidable

Theorem ([Brand & Zafiropulo 1983])

The following problem is undecidable:

**INPUT:** CFM $A$.

**QUESTION:** Is there a some MSC that is accepted by $A$?

Theorem

The following problem is decidable:

**INPUT:** CFM $A$ and integer $B$.

**QUESTION:** Is there an $\exists$-$B$-bounded MSC that is accepted by $A$?
Emptiness for CFMs is undecidable

Theorem ([Brand & Zafiropulo 1983])

The following problem is undecidable:

**INPUT:** CFM $A$.

**QUESTION:** Is there a some MSC that is accepted by $A$?

Theorem

The following problem is decidable:

**INPUT:** CFM $A$ and integer $B$.

**QUESTION:** Is there an $\exists$-$B$-bounded MSC that is accepted by $A$?
Theorem

The following problem is decidable:

**INPUT:** EC-CFM $A$ and an integer $B$.

**QUESTION:** Is there a T-MSC $T$ accepted by $A$ such that $T$ has a $B$-bounded *timed* linearization?

Proof.

Main idea is to construct a timed automaton that accepts precisely the $B$-bounded *timed* linearizations of T-MSCs accepted by $A$:

1. construct an infinite timed automaton by maintaining the partial order “cleverly” along the timed-word run.
2. reduce to finite-state timed automaton by combining guards.

The number of clocks of the timed automaton is $B^{O(|\text{Agents}|^2)}$. 
Theorem

The following problem is decidable:

**INPUT:** EC-CFM $A$ and an integer $B$.

**QUESTION:** Is there a T-MSC $T$ accepted by $A$ such that $T$ has a $B$-bounded *timed* linearization?

Proof.

Main idea is to construct a timed automaton that accepts precisely the $B$-bounded *timed* linearizations of T-MSCs accepted by $A$:

1. construct an infinite timed automaton by maintaining the partial order “cleverly” along the timed-word run.
2. reduce to finite-state timed automaton by combining guards.

The number of clocks of the timed automaton is $B^{O(|Agents|^2)}$. 
Emptiness for EC-CFMs

**Theorem**

The following problem is decidable:

**Input:** EC-CFM $A$ and an integer $B$.

**Question:** Is there a T-MSC $T$ accepted by $A$ such that $T$ has a $B$-bounded timed linearization?

**Corollary**

The following problem is decidable:

**Input:** TMSO formula $\varphi$ and an integer $B$.

**Question:** Is there a T-MSC model $T$ of $\varphi$ such that $T$ has a $B$-bounded timed linearization?
Related work

- [Krcal & Yi, CAV 2006]
  - Timed CFMs
  - (Un)decidability results for one and two channels

- [Chandrasekaran & Mukund, FORMATS 2006]
  - Timed CFMs
  - Modeling channels in UPPAAL

- [S. Akshay & Kumar & Mukund, CONCUR 2007]
  - Timed CFMs and timed HMSCs
  - Matching problems
Future work

- Implementability
  - Timing alphabets that give a decidable architecture
  - Local clocks, universally bounded/existentially bounded
  - Practical implementability – finiteness of clocks

- Other automata models

- Different logics