

Synthesis and Analysis of Timed Communicating Systems

S. Akshay^{1,2} Benedikt Bollig¹ Paul Gastin¹

¹LSV, ENS Cachan, CNRS, France

²Institute of Mathematical Sciences, Chennai, India

Graduiertenkolleg AlgoSyn, Aachen
December 5, 2007

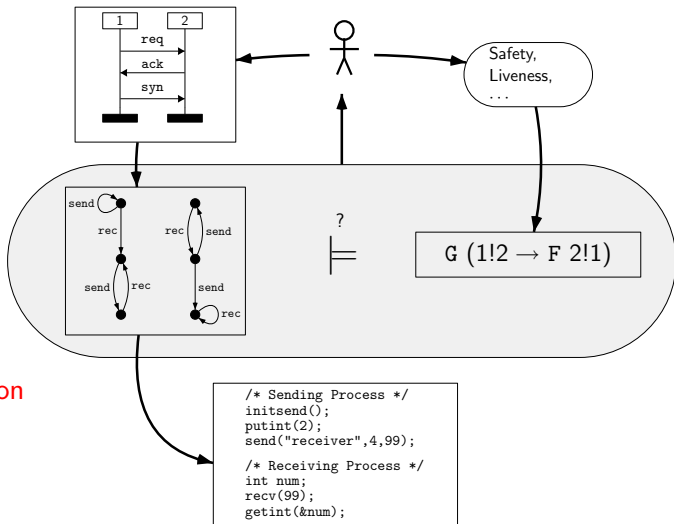
Formal methods

specification

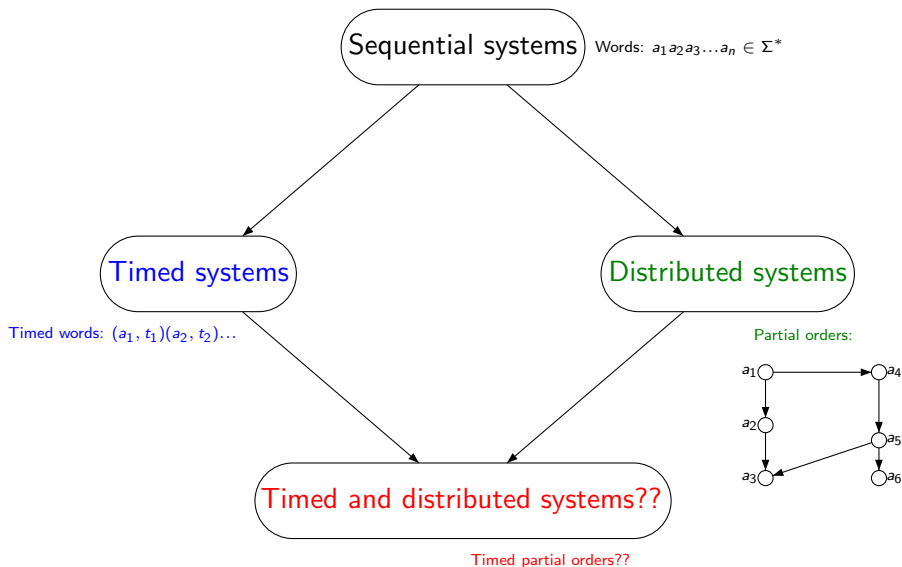
synthesis

modeling &
verification

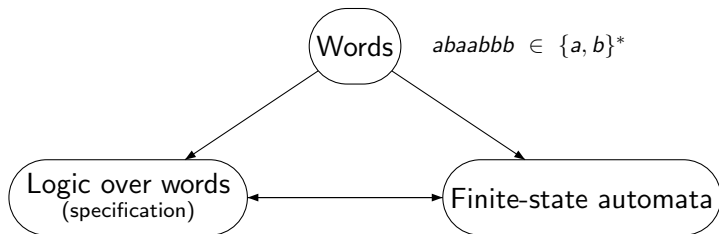
code generation



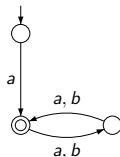
The automata/systems scene



In the beginning there were ...



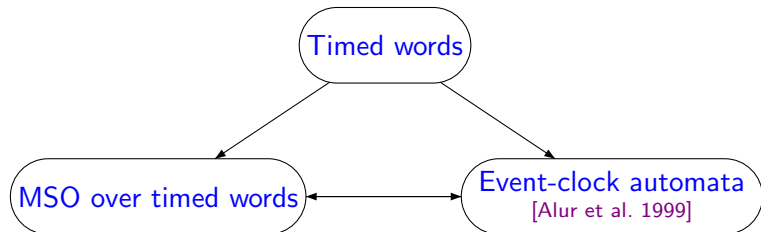
$$\begin{aligned} \exists X \quad & a(x_{min}) \\ \wedge \quad & x_{min} \in X \\ \wedge \quad & x_{max} \in X \\ \wedge \quad & \forall x(x \in X \leftrightarrow \neg(x + 1 \in X)) \end{aligned}$$



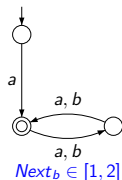
[Büchi, Elgot 1960]

Timed setting

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

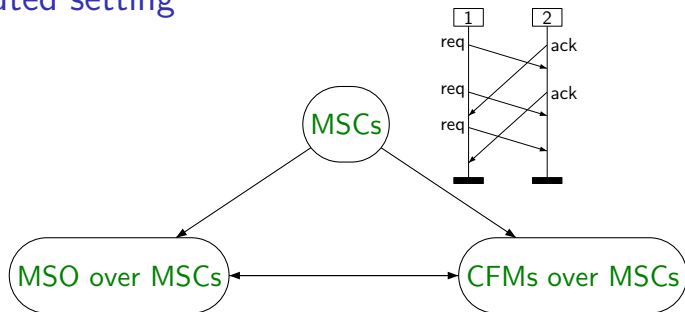


$\exists X$ $a(x_{min})$
 $\wedge x_{min} \in X$
 $\wedge x_{max} \in X$
 $\wedge \forall x(x \in X \leftrightarrow \neg(x + 1 \in X))$
 $\wedge \forall x(\neg x \in X \rightarrow \delta(x, Next_b(x)) \in [1, 2])$

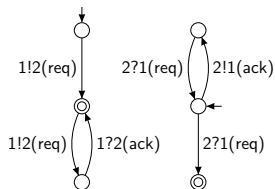


[D'Souza 2003]

Distributed setting



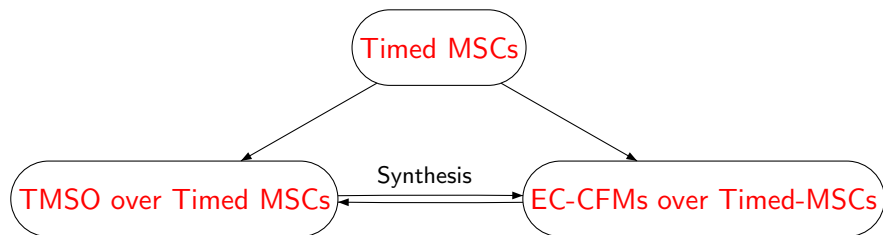
$$\forall x(1!2(\text{req})(x) \rightarrow \exists y(x \leq y \wedge 2!1(\text{ack})(y)))$$



[Genest & Kuske & Muscholl 2004]

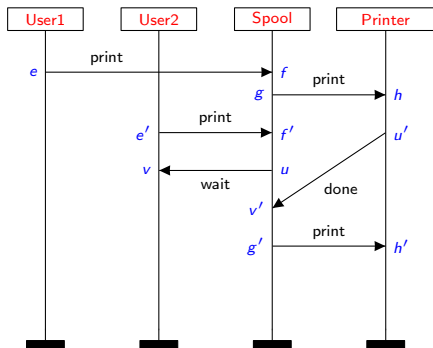
Goal in this work

- 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:



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, ℓ) with

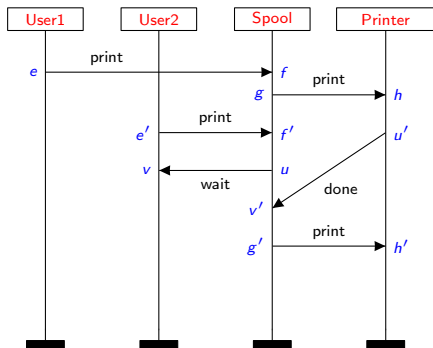
$E = \{e, f, g, u, \dots\}$

$\leq = \{(e, f), (f, u), (e, v), \dots\}$

$\ell : E \rightarrow \{\text{User1!Spool}(\text{print}), \text{Printer?Spool}(\text{print}), \dots\}$

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:

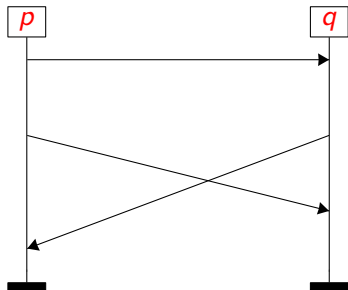
Labelled partial order (E, \leq, ℓ) with

$E = \{e, f, g, u, \dots\}$

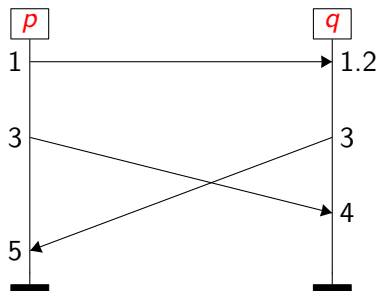
$\leq = \{(e, f), (f, u), (e, v), \dots\}$

$\ell : E \rightarrow \{\mathbf{User1!Spool}(\text{print}), \mathbf{Printer?Spool}(\text{print}), \dots\}$

Introducing timing (T-MSC)

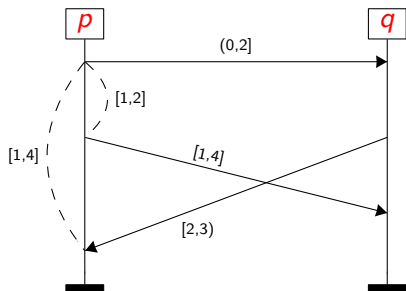


Introducing timing (T-MSC)



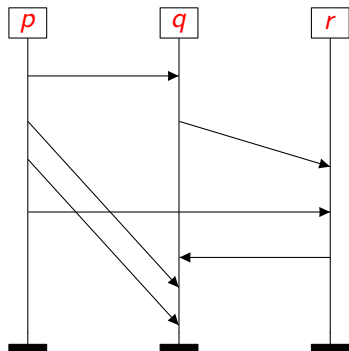
- Attach time stamps (which are non-negative real numbers) to events.
- This is the natural formalism for timing which extends from timed words.
- However, it is not natural for specification by engineers!
So we introduce another model ...

MSC with timing constraints (TC-MSC)

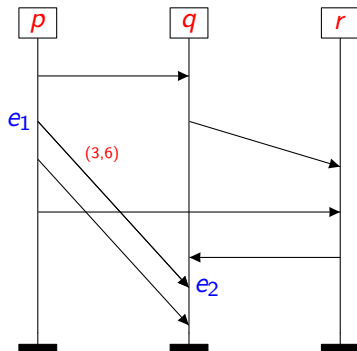


- We attach time intervals to “selected pairs” of events.
- We can restrict the pairs and thus control timing. This also will depend on and determine our logic and automaton models.
- This is natural for specification.

Allowed timing intervals

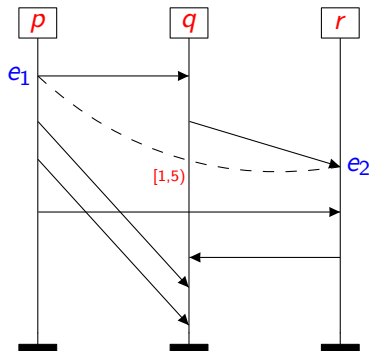


Allowed timing intervals



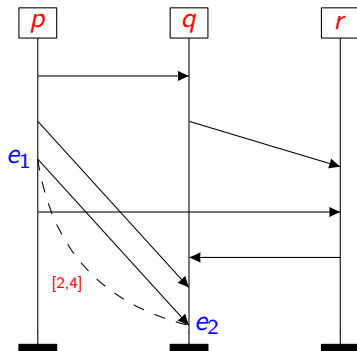
Allows timing between all pairs of events that form **messages**.

Allowed timing intervals



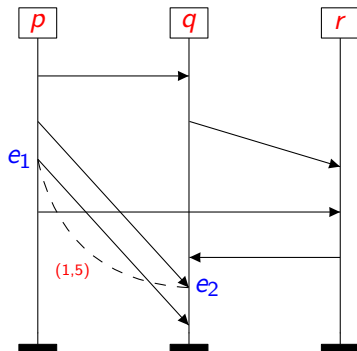
Allows timing between an event and another whose label is the **previous appearance** of that action label from this event.

Allowed timing intervals



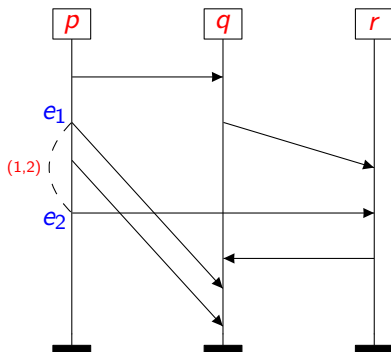
Allows timing between an event and another whose label is the **previous appearance** of that action label from this event.

Allowed timing intervals



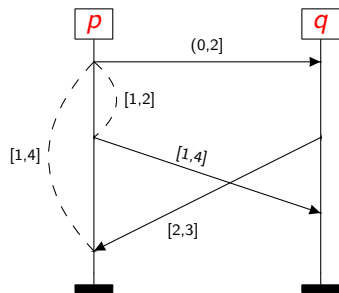
Allows timing between an event and another whose label is the very **next appearance** of that action label.

Allowed timing intervals

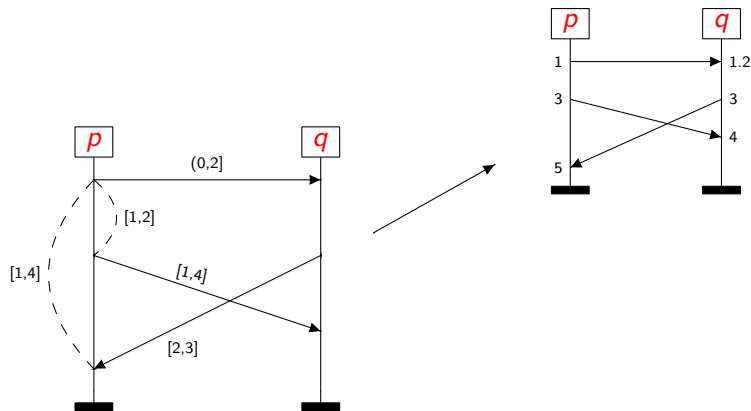


Allows timing between an event and another whose label is the very **next appearance** of that action label.

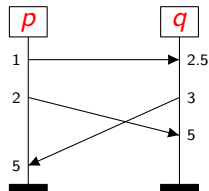
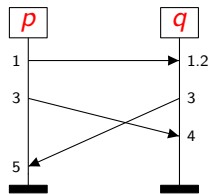
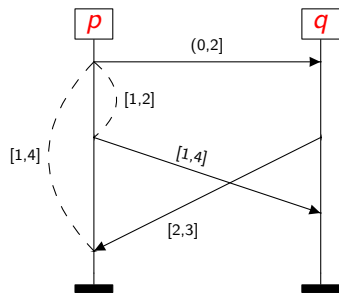
TC-MSC, T-MSC, and timed words



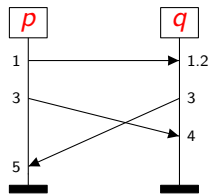
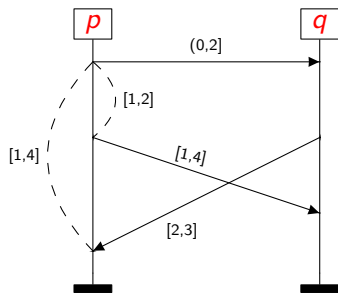
TC-MSC, T-MSC, and timed words



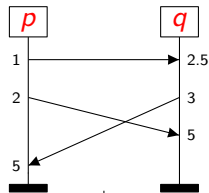
TC-MSC, T-MSC, and timed words



TC-MSC, T-MSC, and timed words

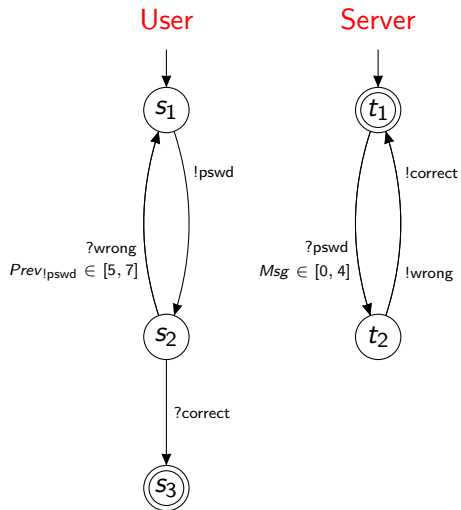


$(p!q, 1)(q!p, 1.2)(p!q, 3)(q!p, 3)(q?p, 4)(p?q, 5)$
 $(p!q, 1)(q!p, 1.2)(q!p, 3)(p!q, 3)(q?p, 4)(p?q, 5)$

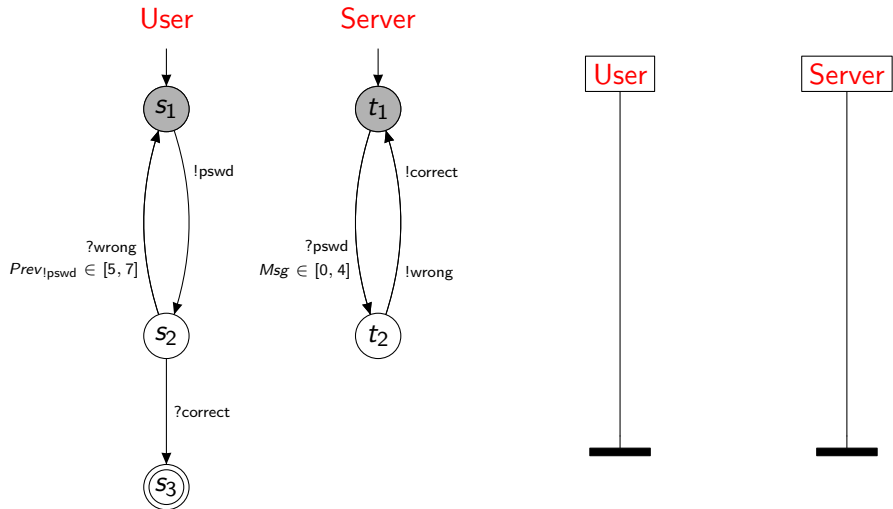


$(p!q, 1)(q!p, 1.2)(p!q, 3)(q!p, 3)(p?q, 5)(q?p, 5)$
 $(p!q, 1)(q!p, 1.2)(p!q, 3)(q!p, 3)(q?p, 5)(p?q, 5)$

Event-clock communicating finite-state machine

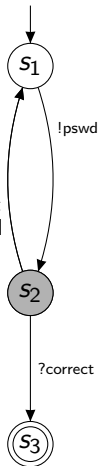


Event-clock communicating finite-state machine

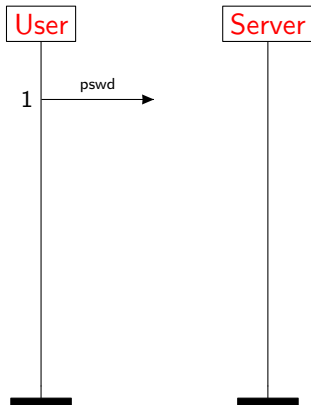
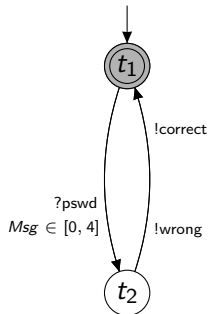


Event-clock communicating finite-state machine

User

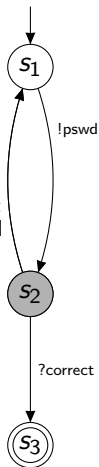


Server

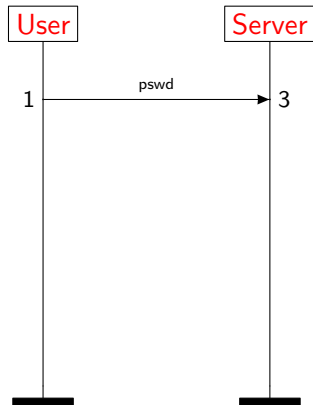
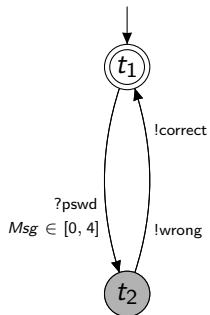


Event-clock communicating finite-state machine

User

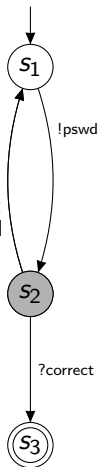


Server

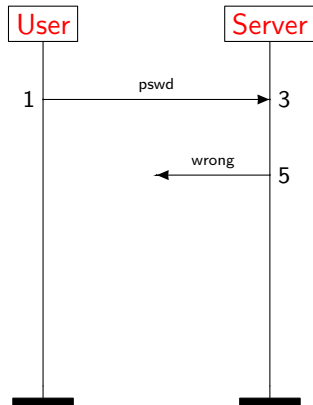
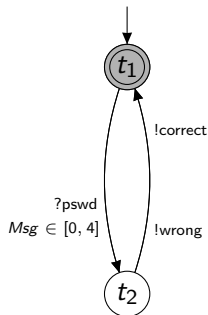


Event-clock communicating finite-state machine

User

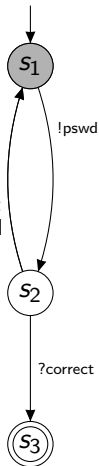


Server

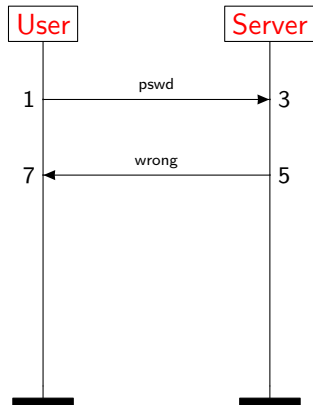
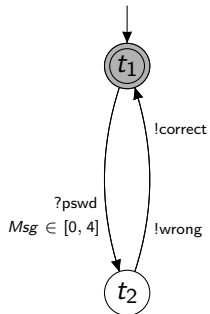


Event-clock communicating finite-state machine

User

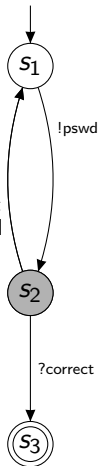


Server

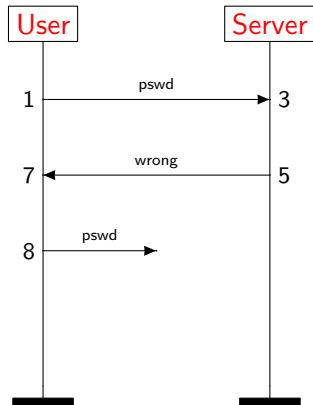
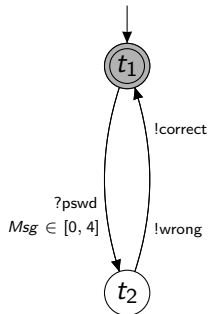


Event-clock communicating finite-state machine

User

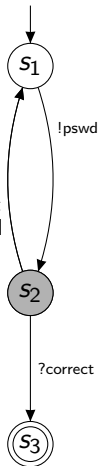


Server

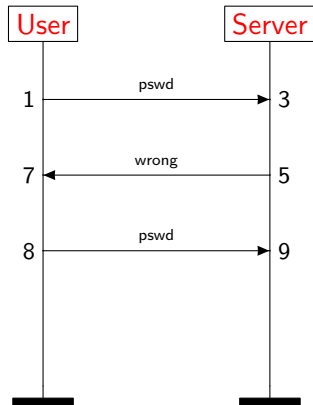
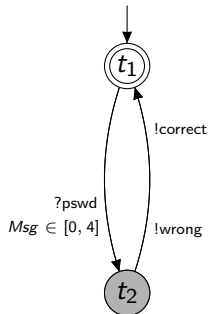


Event-clock communicating finite-state machine

User

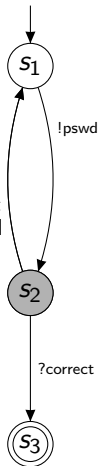


Server

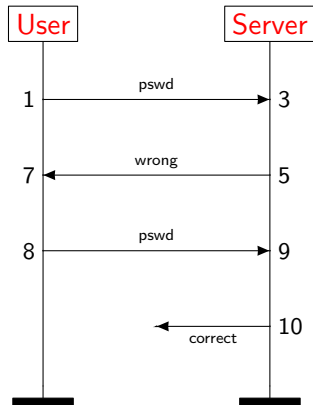
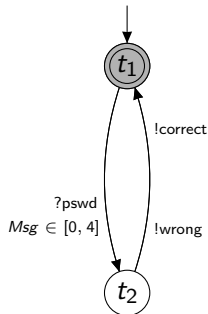


Event-clock communicating finite-state machine

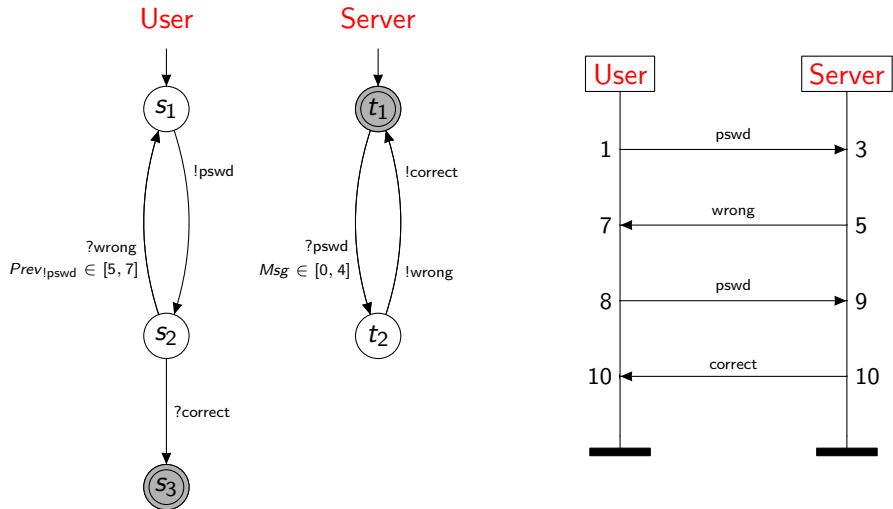
User



Server

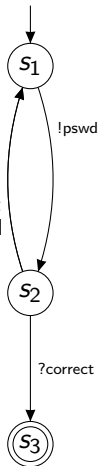


Event-clock communicating finite-state machine

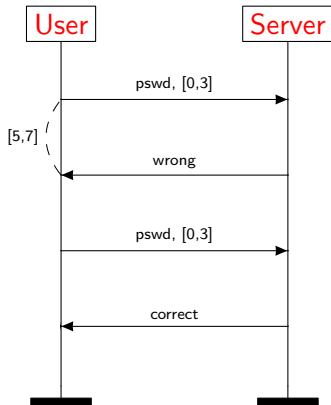
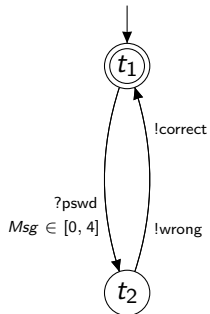


Event-clock communicating finite-state machine

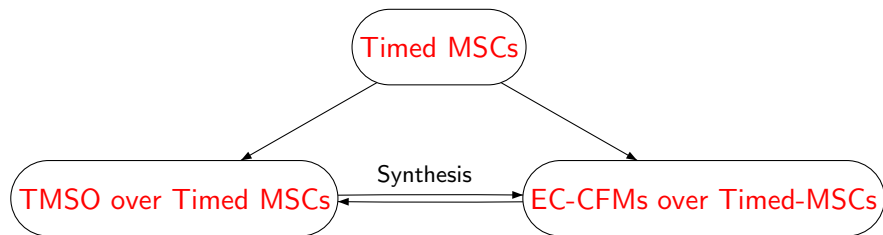
User



Server



Goal in this work



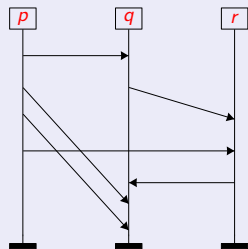
MSO logic vs. automata – The untimed case

Definition

Monadic second-order logic (MSO):

$$\varphi ::= x \leq y \mid x \triangleleft_{\text{proc}} y \mid x \triangleleft_{\text{msg}} y$$

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x \varphi \mid \exists X \varphi$$



$$\models \exists x \exists y (q ? p(x) \wedge r ? p(y) \wedge x \leq y)$$

MSO logic vs. automata – The untimed case

Theorem

Monadic second-order logic (MSO):

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

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi$$

- any implementation has a specification

for any CFM \mathcal{A} , there is an MSO formula φ such that \mathcal{A} recognizes precisely the models of φ

- not every specification is implementable [B. & Leucker 2004]

MSO logic vs. automata – The untimed case

Theorem

Monadic second-order logic (MSO):

$$\varphi ::= x \leq y \mid \cancel{x \leq_{\text{proc}} Y} \mid \cancel{x \leq_{\text{msg}} Y}$$

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x \varphi \mid \exists X \varphi$$

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

MSO logic vs. automata – The untimed case

Theorem

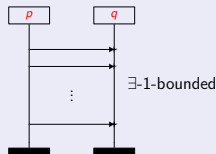
Monadic second-order logic (MSO):

$$\varphi ::= x \leq y \mid x \triangleleft_{\text{proc}} y \mid x \triangleleft_{\text{msg}} y$$

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi$$

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**):

$$\varphi ::= \cancel{x \leq y} \mid x \triangleleft_{\text{proc}} y \mid x \triangleleft_{\text{msg}} y$$

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg \varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x \varphi \mid \exists X \varphi$$

- every implementation has a specification
- every specification is implementable [B. & Leucker 2004]

MSO logic vs. automata – The timed case

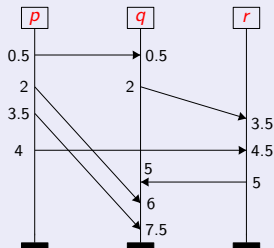
Definition

Timed monadic second-order logic (TMSO):

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

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

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi$$



\models

$$\forall x (p!q(x) \rightarrow \delta(x, \text{Msg}(x)) \in [0, 5]) \\ \wedge \exists x (p!q(x) \wedge \delta(x, \text{Next}_{q?p}(x)) \in [1, 3])$$

MSO logic vs. automata – The timed case

Theorem

Timed monadic second-order logic (TMSO):

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

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

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi$$

If we restrict to \exists -bounded channels:

- every implementation has a specification
- every specification is implementable

MSO logic vs. automata – The timed case

Theorem

Timed monadic second-order logic (TMSO):

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

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

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi$$

More precisely ...

For a set L of *untimed*- \exists - B -bounded T-MSCs, the following are equivalent:

- There is an EC-CFM recognizing L .
- There is a TMSO formula φ defining L .

MSO logic vs. automata – The timed case

Theorem

Timed monadic second-order logic (TMSO):

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

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

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi$$

This is, however, not completely intuitive:

$$\begin{aligned} & \forall x(p!q(x) \vee q?p(x)) \\ \wedge & \forall x(p!q(x) \rightarrow \delta(x, \text{Next}_{p!q}(x)) \in [0, 0] \\ & \wedge \delta(x, \text{Msg}(x)) \in (0, \infty)) \end{aligned}$$

defines an untimed- \exists -1-bounded set of T-MSCs.

MSO logic vs. automata – The timed case

Theorem

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

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

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

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi$$

- every implementation has a specification
- every specification is implementable

MSO logic vs. automata – The timed case

Theorem

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

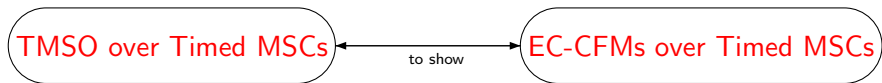
$$\varphi ::= x \leq y \mid x \prec_{\text{proc}} y \mid x \prec_{\text{msg}} y$$

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

$$\text{act}(x) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x\varphi \mid \exists X\varphi$$

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

Sketch of proof



Sketch of proof

TMSO over Timed MSCs

EC-CFMs over Timed MSCs

MSO over MSCs

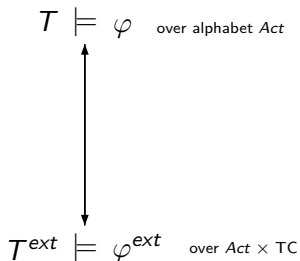
CFMs over MSCs



Sketch of proof

TMSO over Timed MSCs

EC-CFMs over Timed MSCs



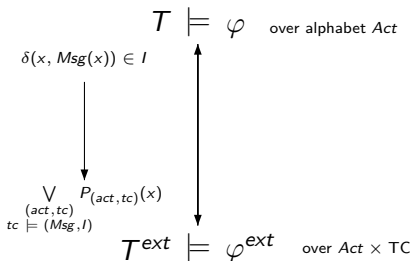
MSO over MSCs

CFMs over MSCs

Sketch of proof

TMSO over Timed MSCs

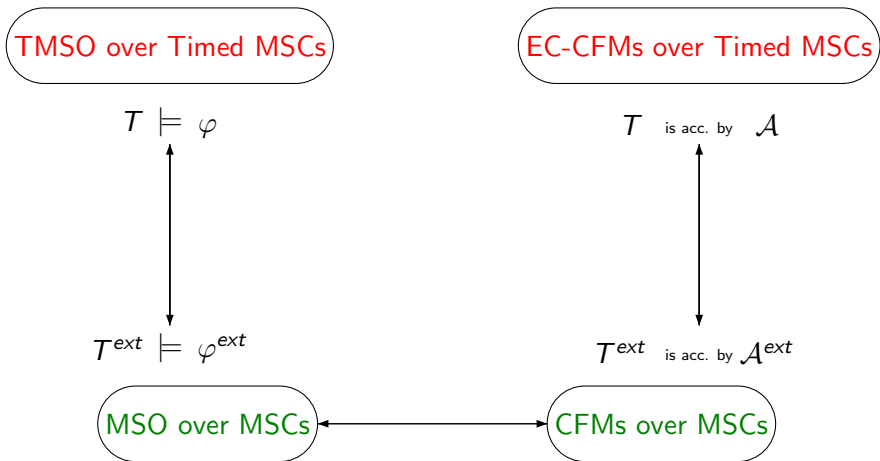
EC-CFMs over Timed MSCs



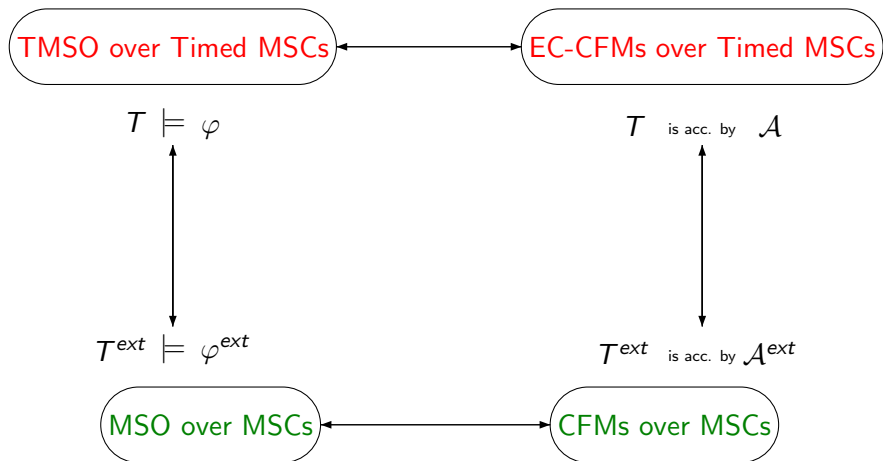
MSO over MSCs

CFMs over MSCs

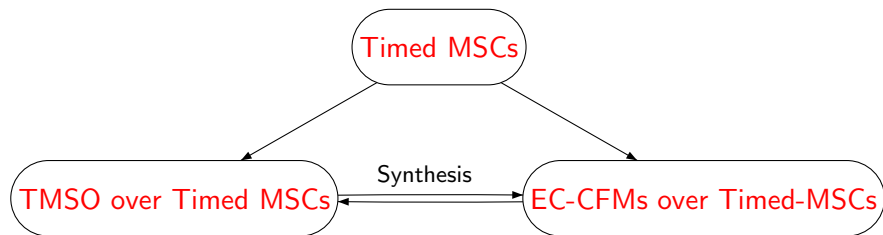
Sketch of proof



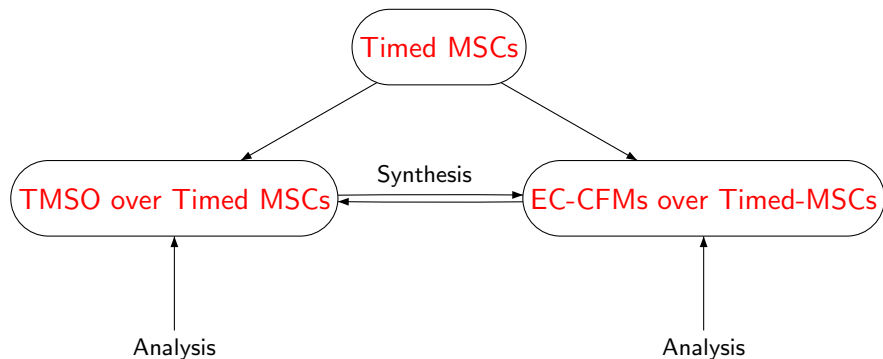
Sketch of proof



Goal in this work



Goal in this work



Emptiness for CFMs is undecidable

Theorem ([Brand & Zafiropulo 1983])

The following problem is undecidable:

INPUT: CFM \mathcal{A} .

QUESTION: Is there a some MSC that is accepted by \mathcal{A} ?

Theorem

The following problem is decidable:

INPUT: CFM \mathcal{A} and integer B .

QUESTION: Is there an \exists - B -bounded MSC that is accepted by \mathcal{A} ?

Emptiness for CFMs is undecidable

Theorem ([Brand & Zafiropulo 1983])

The following problem is undecidable:

INPUT: CFM \mathcal{A} .

QUESTION: Is there a some MSC that is accepted by \mathcal{A} ?

Theorem

The following problem is decidable:

INPUT: CFM \mathcal{A} and integer B .

QUESTION: Is there an \exists - B -bounded MSC that is accepted by \mathcal{A} ?

Emptiness for EC-CFMs

Theorem

The following problem is decidable:

INPUT: EC-CFM \mathcal{A} and an integer B .

QUESTION: Is there a T-MSC T accepted by \mathcal{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 \mathcal{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 $B^{\mathcal{O}(|Agents|^2)}$. □

Emptiness for EC-CFMs

Theorem

The following problem is decidable:

INPUT: EC-CFM \mathcal{A} and an integer B .

QUESTION: Is there a T-MSC T accepted by \mathcal{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 \mathcal{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 $B^{\mathcal{O}(|Agents|^2)}$. □

Emptiness for EC-CFMs

Theorem

The following problem is decidable:

INPUT: EC-CFM \mathcal{A} and an integer B .

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

Corollary

The following problem is decidable:

INPUT: TMSO formula φ and an integer B .

QUESTION: Is there a T-MSC model T of φ 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