

Unreliability in Practical Subclasses of Communicating Systems (FSTTCS25)

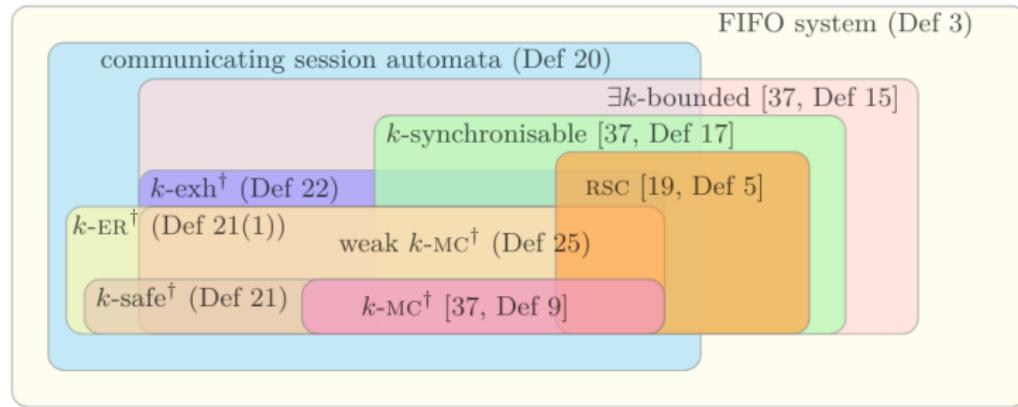
Amrita Suresh, Nobuko Yoshida, University of Oxford, UK
Presented by: Gabriele Genovese

13/11/2025

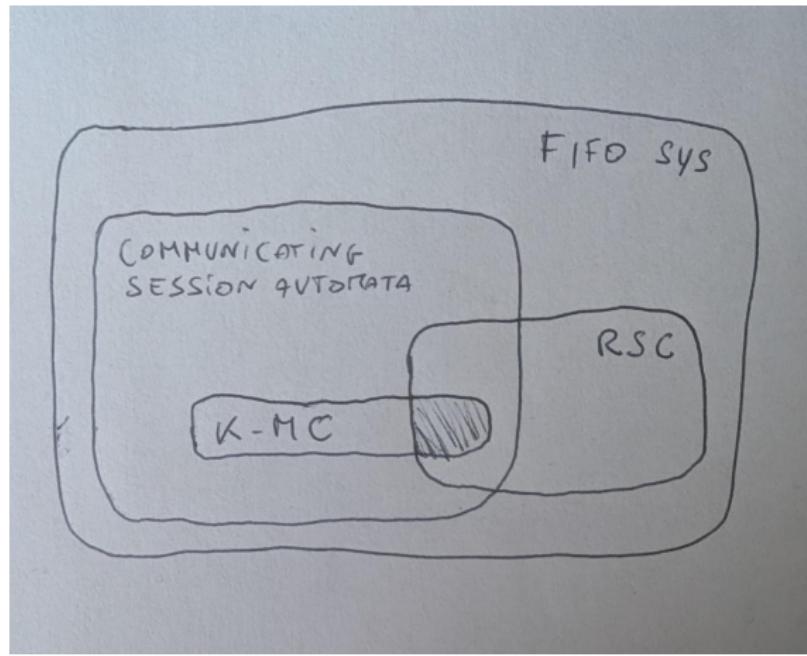
Introduction

- ▶ FIFO system → Turing complete
- ▶ Perfect channels often assumed
- ▶ Aim: study unreliability...
 - ▶ interferences
 - ▶ crash of processes
- ▶ ...in two Practical Subclasses of FIFO (half-duplex) system:
 - ▶ Realisable with Synchronous Communication (RSC)
 - ▶ k -multiparty compatibility (k -MC)

Classes of communication systems



Classes of communication systems (simplified)



Aim

Do inclusion (or membership) of these subclasses remains decidable?

Do the complexity remains the same?

Can we translate the results in the MPST world?

Aim - Spoiler

Do inclusion (or membership) of these subclasses remains decidable? **Yes.**

Do the complexity remains the same? **Yes.**

Can we translate the results in the MPST world? **Yes.**

Contributions

- ▶ *i*-RSC and *weak k*-MC with *interferences* is decidable
- ▶ *i*-RSC and *weak k*-MC with *crash failures* is decidable
- ▶ *Translation* from local types (MPST) to crash-handling FIFO systems with proof of preserving trace semantics
- ▶ Evaluation of protocols with tools

Preliminaries

► **Definition 2** (FIFO automaton). A FIFO automaton \mathcal{A}_p , associated with p , is defined as $\mathcal{A}_p = (Q_p, \delta_p, q_{0p})$ where: Q_p is the finite set of control-states, $\delta_p \subseteq Q_p \times \text{Act} \times Q_p$ is the transition relation, and $q_{0p} \in Q_p$ is the initial control-state.

► **Definition 3** (FIFO system). A FIFO system $\mathcal{S} = (\mathcal{A}_p)_{p \in \mathbb{P}}$ is a set of communicating FIFO automata. A configuration of \mathcal{S} is a pair $\gamma = (\vec{q}; \vec{w})$ where $\vec{q} = (q_p)_{p \in \mathbb{P}}$ is called the global state with $q_p \in Q_p$ being one of the local control-states of \mathcal{A}_p , and where $\vec{w} = (w_{pq})_{pq \in Ch}$ with $w_{pq} \in \Sigma^*$.

Interferences (\succeq)

Reflexivity

$$\frac{a \in \Sigma}{a \succeq a}$$

Transitivity

$$\frac{w \succeq w' \quad w' \succeq w''}{w \succeq w''}$$

Additivity

$$\frac{w_1 \succeq w'_1 \quad w_2 \succeq w'_2}{w_1 \cdot w_2 \succeq w'_1 \cdot w'_2}$$

Integrity

$$\frac{\varepsilon \succeq w}{w = \varepsilon}$$

Non-expansion

$$\frac{w \succeq w'}{|w| \geq |w'|}$$

Interferences (\succeq)

Reflexivity

$$\frac{a \in \Sigma}{a \succeq a}$$

Transitivity

$$\frac{w \succeq w' \quad w' \succeq w''}{w \succeq w''}$$

Additivity

$$\frac{w_1 \succeq w'_1 \quad w_2 \succeq w'_2}{w_1 \cdot w_2 \succeq w'_1 \cdot w'_2}$$

Integrity

$$\frac{\varepsilon \succeq w}{w = \varepsilon}$$

Non-expansion

$$\frac{w \succeq w'}{|w| \geq |w'|}$$

- ▶ Additivity: failures can happen at any part of the words

Interferences (\succeq)

Reflexivity

$$\frac{a \in \Sigma}{a \succeq a}$$

Transitivity

$$\frac{w \succeq w' \quad w' \succeq w''}{w \succeq w''}$$

Additivity

$$\frac{w_1 \succeq w'_1 \quad w_2 \succeq w'_2}{w_1 \cdot w_2 \succeq w'_1 \cdot w'_2}$$

Integrity

$$\frac{\varepsilon \succeq w}{w = \varepsilon}$$

Non-expansion

$$\frac{w \succeq w'}{|w| \geq |w'|}$$

- ▶ Additivity: failures can happen at any part of the words
- ▶ Integrity: ϵ is the least word

Interferences (\succeq)

<i>Reflexivity</i>	<i>Transitivity</i>	<i>Additivity</i>	<i>Integrity</i>	<i>Non-expansion</i>
$\frac{a \in \Sigma}{a \succeq a}$	$\frac{w \succeq w' \quad w' \succeq w''}{w \succeq w''}$	$\frac{w_1 \succeq w'_1 \quad w_2 \succeq w'_2}{w_1 \cdot w_2 \succeq w'_1 \cdot w'_2}$	$\frac{\varepsilon \succeq w}{w = \varepsilon}$	$\frac{w \succeq w'}{ w \geq w' }$

- ▶ Additivity: failures can happen at any part of the words
- ▶ Integrity: ϵ is the least word
- ▶ Non-expansion: \succeq preserves the size of words

Type of interferences

Type of interferences

- ▶ *Lossiness*: message is lost during transmission ($a \succeq \epsilon$)

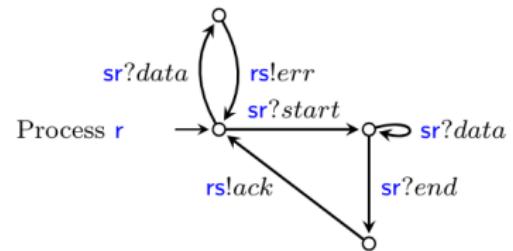
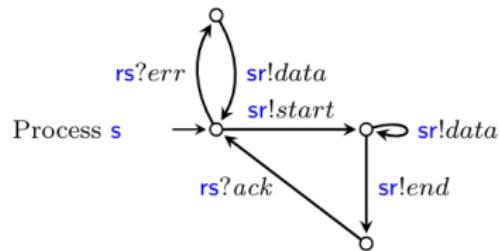
Type of interferences

- ▶ *Lossiness*: message is lost during transmission ($a \succeq \epsilon$)
- ▶ *Corruption*: message is transformed during transmission ($a \succeq b$)

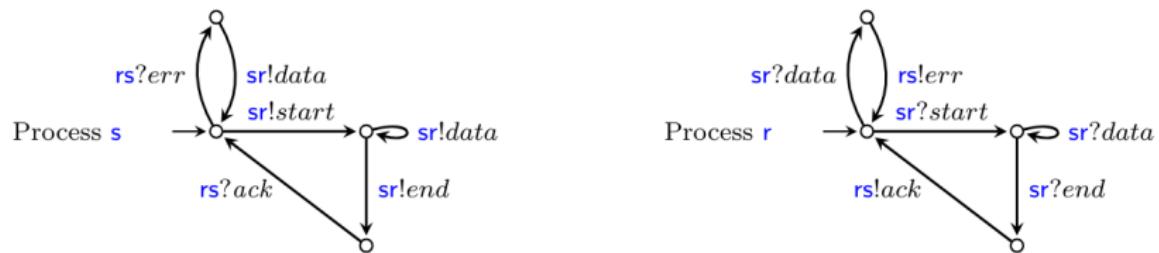
Type of interferences

- ▶ *Lossiness*: message is lost during transmission ($a \succeq \epsilon$)
- ▶ *Corruption*: message is transformed during transmission ($a \succeq b$)
- ▶ *Out-of-order*: messages arrives at different time ($a \cdot b \succeq b \cdot a$)

Example

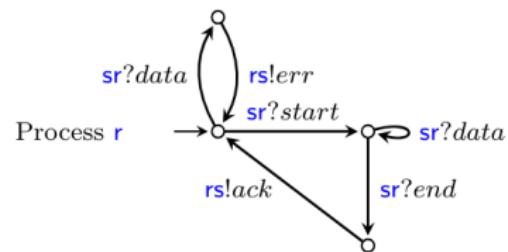
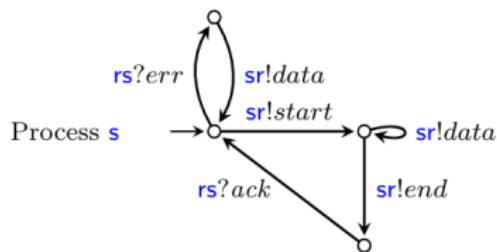


Example



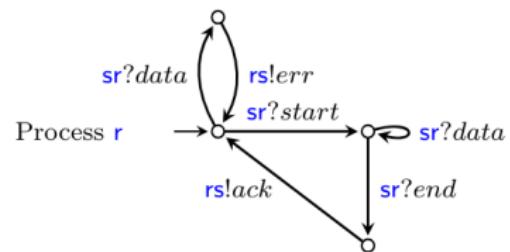
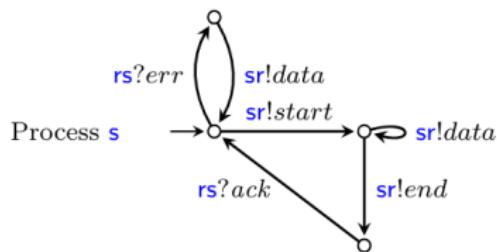
- ▶ Corruption: $sr!start.sr?start.sr!data.sr?end.rs!ack.sr!data.$

Example



- ▶ Corruption: $sr!start.sr?start.sr!data.sr?end.rs!ack.sr!data.$
- ▶ Lossiness: $sr!start.sr?start.sr!data.sr?data.sr!end$

Example



- ▶ Corruption: $sr!start.sr?start.sr!data.sr?end.rs!ack.sr!data.$
- ▶ Lossiness: $sr!start.sr?start.sr!data.sr?data.sr!end$
- ▶ Out-of-order: $sr!start.sr?start.sr!data.sr!end.sr?end.rs!ack.$
 $rs?ack.sr?data.rs!err.rs?err$

- ▶ Extend the definition of *matching pairs* to include interference:
 - ▶ the message can be different (corruption)
 - ▶ a “receive” action is not strictly after a “send” action (out-of order)
- ▶ An *interaction* is either a matching pair, or a singleton (lossiness)

► **Definition 7** (Matching pair with interference). *Given an execution $e = a_1 \dots a_n$, if there exists a channel pq , messages $m, m' \in \Sigma$ and $j, j', k, k' \in \{1, \dots, n\}$ where $j < j'$, and the following four conditions:*

(1) $a_j = \text{pq}!m$; (2) $a_{j'} = \text{pq}?m'$; (3) a_j is the k -th send action to pq in e ; and (4) $a_{j'}$ is the k' -th receive action on pq in e , then we say that $\{j, j'\} \subseteq \{1, \dots, n\}$ is a matching pair with interference, or i -matching pair.

► **Definition 9** (Interaction). *An interaction of e is either a (perfect or i -) matching pair, or a singleton $\{j\}$ such that a_j is a send action and j does not belong to any matching pair (such an interaction is called unmatched send).*

Example

$$e = a_1 \dots a_5 = pq!a \cdot qp!b \cdot qp?b \cdot pq!c \cdot pq?c$$

Two *valid* communication:

$$\text{Comm}(e) = \{\nu_1, \nu_2\}$$

Example

$$e = a_1 \dots a_5 = pq!a \cdot qp!b \cdot qp?b \cdot pq!c \cdot pq?c$$

Two *valid* communication:

- ▶ $\nu_1 = \{\{1, 5\}, \{2, 3\}, \{4\}\}$

$$\text{Comm}(e) = \{\nu_1, \nu_2\}$$

Example

$$e = a_1 \dots a_5 = pq!a \cdot qp!b \cdot qp?b \cdot pq!c \cdot pq?c$$

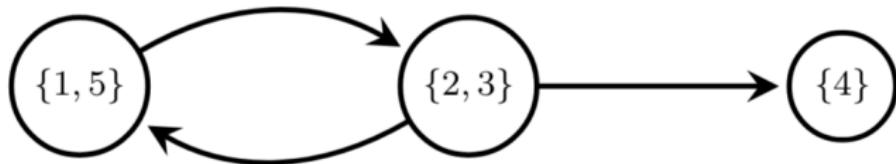
Two *valid* communication:

- ▶ $\nu_1 = \{\{1, 5\}, \{2, 3\}, \{4\}\}$
- ▶ $\nu_2 = \{\{1\}, \{2, 3\}, \{4, 5\}\}$

$$\text{Comm}(e) = \{\nu_1, \nu_2\}$$

Conflict graph and borderline violation

- ▶ Characterise causally equivalent executions, using conflict graph



- ▶ *Borderline violation*: “minimal counter-example” for non-RSC behaviour (regular language)

Conflict graph and borderline violation

Definition 11 (Conflict graph). Given an execution (e, ν) , the conflict graph $\text{cgraph}(e, \nu)$ the directed graph $(\nu, \rightarrow_{e, \nu})$ where for all interactions $\chi_1, \chi_2 \in \nu$, $\chi_1 \rightarrow_{e, \nu} \chi_2$ if there is $j_1 \in \chi_1$ and $j_2 \in \chi_2$ such that $j_1 \prec_{e, \nu} j_2$.

► **Definition 15** (Borderline violation). An execution (e, ν) is a borderline violation if (1) (e, ν) is not causally equivalent to an i -RSC execution, (2) $e = e' \cdot \text{c?}m$ for some execution e' such that (a) for all $\nu' \in \text{Comm}(e')$, (e', ν') is equivalent to an i -RSC execution and (b) there exists $\nu_1 \in \text{Comm}(e')$ such that (e', ν_1) is an i -RSC execution.

All matching pairs in valid communication are of the form $\{j, j + 1\}$.

► **Definition 12** (*i*-RSC system). An execution (e, ν) is *i*-RSC if all matching pairs in ν are of the form $\{j, j + 1\}$. A system \mathcal{S} is *i*-RSC if for all tuples (e, ν) such that $e \in \text{executions}(\mathcal{S})$ and $\nu \in \text{Comm}(e)$, we have $\text{cgraph}(e, \nu) = \text{cgraph}(e', \nu')$ where (e', ν') is an *i*-RSC execution.

ν_2 is an *i*-RSC execution.

► **Theorem 19.** Given a system \mathcal{S} of size n , deciding whether it is an *i*-RSC system can be done in time $\mathcal{O}(n^{|\mathbb{P}|+2} |Ch|^5 \times 2^{|Ch|} \times |\Sigma|^2)$.

Lemmas

An execution is an i -RSC execution iff the conflict graph is acyclic:

- **Lemma 14.** An execution (e, ν) is causally equivalent to an i -RSC execution iff the associated conflict graph $\text{cgraph}(e, \nu)$ is acyclic.

A system is i -RSC iff every execution is not a borderline violation

- **Lemma 16.** \mathcal{S} is i -RSC if and only if for all $e \in \text{executions}(\mathcal{S})$ and $\nu \in \text{Comm}(e)$, (e, ν) is not a borderline violation.

The language of borderline violation is regular:

- **Lemma 17.** Let \mathcal{S} with $\text{product}(\mathcal{S}) = (Q, \Sigma, Ch, \text{Act}, \delta, q_0)$. There is a non-deterministic finite state automaton \mathcal{A}_{bv} computable in time $\mathcal{O}(|Ch|^3|\Sigma|^2)$ such that $\mathcal{L}(\mathcal{A}_{bv}) = \{e \in \text{Act}_{nr}^*.\text{Act}? \mid \exists \nu \in \text{Comm}(e) \text{ such that } (e, \nu) \text{ is a borderline violation}\}$.

The subset of executions that begin with an i -RSC prefix and terminate with a reception is regular:

- **Lemma 18.** Let \mathcal{S} be a FIFO system. There exists a non-deterministic finite state automaton \mathcal{A}_{rsc} over $\text{Act}_{nr} \cup \text{Act?}$ such that $\mathcal{L}(\mathcal{A}_{rsc}) = \{e \cdot \text{pq}?m \in \text{Act}_{nr}^*.\text{Act}? \mid e \cdot \text{pq}?m \in \text{executions}(\mathcal{S}) \text{ and } \exists \nu \in \text{Comm}(e) \text{ such that } (e, \nu) \text{ is an } i\text{-RSC execution}\}$, which can be constructed in time $\mathcal{O}(n^{|\mathbb{P}|+2}|Ch|^2 \times 2^{|Ch|})$, where n is the size of \mathcal{S} .

k -Multiparty Compatibility

Definition of k -MC with two properties:

k -Multiparty Compatibility

Definition of k -MC with two properties:

- ▶ k -safety:

k -Multiparty Compatibility

Definition of k -MC with two properties:

- ▶ k -safety:
- ▶ k -ER: **eventual reception**

k -Multiparty Compatibility

Definition of k -MC with two properties:

- ▶ k -safety:
 - ▶ k -ER: **eventual reception**
 - ▶ k -PG: **progress**

k -Multiparty Compatibility

Definition of k -MC with two properties:

- ▶ k -safety:
 - ▶ k -ER: **eventual reception**
 - ▶ k -PG: **progress**
- ▶ k -exhaustivity: all k -reachable configurations, whenever a send action is enabled, it can be fired within a k -bounded execution

Weak k -MC with interferences is decidable

- ▶ k -safety is a too **strong** condition: *remove* progress because in case of *lossiness* it cannot be guaranteed.

Weak k -MC with interferences is decidable

- ▶ k -safety is a too **strong** condition: *remove progress* because in case of *lossiness* it cannot be guaranteed.
- ▶ Definition of **weak** k -MC: A communicating system is weakly k -mc, if it satisfies k -ER and is k -exhaustive.

Weak k -MC with interferences is decidable

- ▶ k -safety is a too **strong** condition: *remove progress* because in case of *lossiness* it cannot be guaranteed.
- ▶ Definition of **weak** k -MC: A communicating system is weakly k -mc, if it satisfies k -ER and is k -exhaustive.
- ▶ Thm: given a system with interference, checking the weak k -mc property is decidable and PSPACE-complete.

FIFO system with crash

- ▶ Declare a set of *reliable* processes \mathcal{R} (can be empty).

FIFO system with crash

- ▶ Declare a set of *reliable* processes \mathcal{R} (can be empty).
- ▶ Not reliable processes can crash

FIFO system with crash

- ▶ Declare a set of *reliable* processes \mathcal{R} (can be empty).
- ▶ Not reliable processes can crash
 - ▶ send a **broadcast** when crash

FIFO system with crash

- ▶ Declare a set of *reliable* processes \mathcal{R} (can be empty).
- ▶ Not reliable processes can crash
 - ▶ send a **broadcast** when crash
- ▶ A process has crash-handling behaviour if it has:

FIFO system with crash

- ▶ Declare a set of *reliable* processes \mathcal{R} (can be empty).
- ▶ Not reliable processes can crash
 - ▶ send a **broadcast** when crash
- ▶ A process has crash-handling behaviour if it has:
 - ▶ Crash handling (CH): no deadlocked processes

FIFO system with crash

- ▶ Declare a set of *reliable* processes \mathcal{R} (can be empty).
- ▶ Not reliable processes can crash
 - ▶ send a **broadcast** when crash
- ▶ A process has crash-handling behaviour if it has:
 - ▶ Crash handling (CH): no deadlocked processes
 - ▶ Crash broadcast (CB): if crash, send broadcast

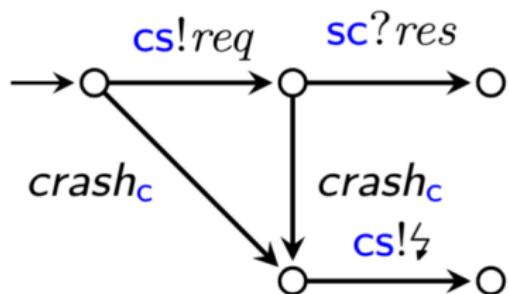
FIFO system with crash

- ▶ Declare a set of *reliable* processes \mathcal{R} (can be empty).
- ▶ Not reliable processes can crash
 - ▶ send a **broadcast** when crash
- ▶ A process has crash-handling behaviour if it has:
 - ▶ Crash handling (CH): no deadlocked processes
 - ▶ Crash broadcast (CB): if crash, send broadcast
 - ▶ Crash redundancy (CR): empty channels

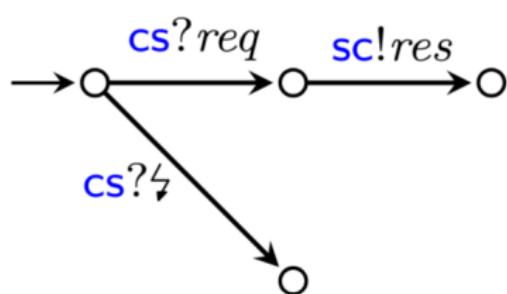
Example of crash handling behaviour

Server is reliable, *client* is not.

Process **c**



Process **s**



Results for RSC and weak k -MC

- ▶ Lem: checking crash-handling is decidable.

Results for RSC and weak k -MC

- ▶ Lem: checking crash-handling is decidable.
- ▶ Lem: boundedness problem remains undecidable in crash-handling systems.

Results for RSC and weak k -MC

- ▶ Lem: checking crash-handling is decidable.
- ▶ Lem: boundedness problem remains undecidable in crash-handling systems.
- ▶ **Thm:** checking inclusion to the RSC class is decidable in crash-handling systems.
 - ▶ Proof adapted from “Cinzia Di Giusto, Loïc Germerie Guizouarn, and Etienne Lozes. Multiparty half-duplex systems and synchronous communications”.

Results for RSC and weak k -MC

- ▶ Lem: checking crash-handling is decidable.
- ▶ Lem: boundedness problem remains undecidable in crash-handling systems.
- ▶ **Thm:** checking inclusion to the RSC class is decidable in crash-handling systems.
 - ▶ Proof adapted from “Cinzia Di Giusto, Loïc Germerie Guizouarn, and Etienne Lozes. Multiparty half-duplex systems and synchronous communications”.
- ▶ **Thm:** checking weak k -MC is PSPACE for crash-handling system generated from communicating session automata.

Local Types with crash-handling

- ▶ **stop** type to denote crashed processes
- ▶ crash-handling branch (**catch**) in an external choice branch

$$\begin{array}{ll} S, T ::= & \text{p?}\{m_i.T_i\}_{i \in I} \mid \text{p!}\{m_i.T_i\}_{i \in I} \\ & \mid \mu t.T \mid t \mid \text{end} \mid \text{stop} \end{array} \quad \begin{array}{l} \text{(external choice, internal choice)} \\ \text{(recursion, type variable, end, crash)} \end{array}$$

LTS over Local Type

► **Definition 34 (LTS over local types).** The relation $T \xrightarrow{a} T'$ for the local type of role p is defined as:

$$[\text{LR1}] \quad q \dagger \{m_i.T_i\}_{i \in I} \xrightarrow{\text{pq} \dagger m_k} T_k, \quad \text{where } \dagger \in \{!, ?\} \text{ and } m_k \neq \text{catch}.$$

$$[\text{LR2}] \quad T[\mu t.T/t] \xrightarrow{a} T' \xrightarrow{\mu t.T \xrightarrow{a} T'} T'$$

$$[\text{LR3}] \quad q \dagger \{m_i.T_i\}_{i \in I} \xrightarrow{\text{crash-broadcast}_p(\dagger)} \text{stop}, \quad \text{where } \dagger \in \{!, ?\}.$$

$$[\text{LR4}] \quad q? \{m_i.T_i\}_{i \in I} \xrightarrow{\text{qp}? \dagger} T_k, \quad \text{if } m_k = \text{catch}.$$

$$[\text{LR5}] \quad T \xrightarrow{\text{qp}? \dagger} T, \quad \forall q \in \mathbb{P} \setminus \{p\} \text{ for } T \in \{\text{stop, end}\}.$$

LTS over Local Type

- LR1 and [LR2] are standard output/input and recursion rules,
- LR3 aid the crash of a process (CB),
- LR4 main rule to enter crash-handling branch (CH),
- LR5 read all dangling crash messages (CR).

Translation from Local Types to FIFO automata

► **Definition 37** (From local types to FIFO automata). Let $\textcolor{violet}{T}_0$ be the local type of participant

p. The automaton corresponding to $\textcolor{violet}{T}_0$ is $\mathcal{A}(\textcolor{violet}{T}_0) = (Q, \delta, q_0)$ where:

1. $Q = \{T' \mid T' \in \textcolor{violet}{T}_0, T' \neq \mathbf{t}, T' \neq \mu \mathbf{t}. T\} \cup \{q_{\text{crash}}\} \cup \{q_{\text{send},r} \mid r \in \mathbb{P} \setminus \{\textcolor{blue}{p}\}\}$
2. $q_0 = \text{strip}(\textcolor{violet}{T}_0)$;
3. δ is the smallest set of transitions such that $\forall T \in Q$:

a. If $T = \mathbf{q} \dagger \{m_i.T_i\}_{i \in I}$ and $k \in I$, $m_k \neq \mathbf{catch}$, and $\dagger \in \{!, ?\}$

$$\left\{ \begin{array}{ll} (T, \mathbf{p} \mathbf{q} \dagger m_k, \text{strip}(T_k)) \in \delta & \text{if } T_k \neq \mathbf{t} \\ (T, \mathbf{p} \mathbf{q} \dagger m_k, \text{strip}(T')) \in \delta & \text{if } T_k = \mathbf{t} \text{ with } \mu \mathbf{t}. T' \in \textcolor{violet}{T}_0. \end{array} \right.$$

b. If $T = \mathbf{q} ? \{m_i.T_i\}_{i \in I}$ with $k \in I$, $m_k = \mathbf{catch}$

$$\left\{ \begin{array}{ll} (T, \mathbf{q} \mathbf{p} ? \dagger, \text{strip}(T_k)) \in \delta & \text{if } T_k \neq \mathbf{t} \\ (T, \mathbf{q} \mathbf{p} ? \dagger, \text{strip}(T')) \in \delta & \text{if } T_k = \mathbf{t} \text{ with } \mu \mathbf{t}. T' \in \textcolor{violet}{T}_0. \end{array} \right.$$

c. If $T \notin \{\mathbf{stop}, \mathbf{end}\}$, then $(T, \text{crash-broadcast}_{\textcolor{blue}{p}}(\dagger), \mathbf{stop}) \subseteq \delta$ where

i. $(T, \text{crash}, q_{\text{crash}}) \in \delta$

ii. $(q_{\text{crash}}, \mathbf{p} \mathbf{r}_1 ! \dagger, q_{\text{send},r_1}) \in \delta$

iii. $(q_{\text{send},r_i}, \mathbf{p} \mathbf{r}_{i+1} ! \dagger, q_{\text{send},r_{i+1}}) \in \delta \quad \forall i \in \{1, \dots, n-2\}, \text{ where } n = |\text{Ch}_{o,\textcolor{blue}{p}}|$

iv. $(q_{\text{send},r_{n-1}}, \text{crash}, \mathbf{stop}) \in \delta$

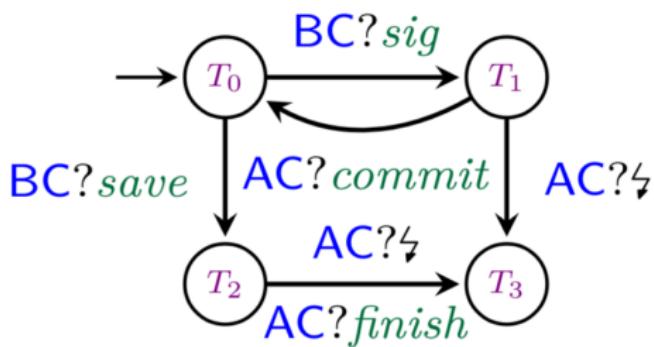
d. If $T \in \{\mathbf{stop}, \mathbf{end}\}$, then $(T, \mathbf{q} \mathbf{p} ? \dagger, T) \in \delta$ for all $\mathbf{q} \in \mathbb{P} \setminus \{\textcolor{blue}{p}\}$.

where $\text{strip}(T) \stackrel{\text{def}}{=} \text{strip}(T')$ if $T = \mu \mathbf{t}. T'$; otherwise $\text{strip}(T) \stackrel{\text{def}}{=} T$.

Example of Local Type with crash-handling

► **Example 36.** Let $\mathbb{P} = \{A, B, C\}$ and $\mathcal{R} = \{B, C\}$. Consider a local type of C : $T = \mu t. B? \{ sig.A? \{ commit.t, catch.end \}, save.A? \{ finish.end, catch.end \} \}$.

Then, the set of all $T' \in T$ is $\{T, B? \{ sig.A? \{ commit.t, catch.end \}, A? \{ commit.t \}, B? \{ save.A? \{ finish.end, catch.end \} \}, A? \{ catch.end \}, A? \{ finish.end \}, end, t \}$.



Results for Local Types

► **Lemma 39.** Assume T_p is a local type. Then $\mathcal{A}(T_p)$ is deterministic, directed and has no mixed states. Moreover, $T_p \approx \mathcal{A}(T_p)$, i.e. $\forall \phi, \phi \in \text{executions}(T_p) \Leftrightarrow \phi \in \text{executions}(\mathcal{A}(T_p))$.

► **Theorem 40.** The FIFO system generated from the translation of crash-stop session types is a crash-handling system. Moreover, it is decidable to check inclusion to the RSC and k -WMC classes.

Experimental evaluation

Tools used and characteristics:

- ▶ RSC-checker: ReSCu
- ▶ k -mc-checker: kmc (added out-of-order)
- ▶ lossiness modelled with self-loops in automata
- ▶ corruption modelled with sending of arbitrary messages
- ▶ examples taken from referenced paper
- ▶ added a test for the Paxos algorithm

Table

Protocol	No errors		Out of order		Lossiness			Corruption				
	<i>k</i> -MC	RSC	<i>k</i> -MC	RSC	<i>k</i> -exh	<i>k</i> -ER	<i>k</i> -PG	RSC	<i>k</i> -exh	<i>k</i> -ER	<i>k</i> -PG	RSC
Alternating Bit [43]	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	no	yes
Alternating Bit [7]	yes	no	yes	yes	yes	yes	no	yes	yes	yes	no	yes
Bargain [36]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Client-Server-Logger [37]	yes	no	yes	no	yes	yes	no	yes	yes	yes	no	yes
Cloud System v4 [27]	yes	yes	yes	no	no	no	no	yes	no	no	no	no
Commit protocol [11]	yes	yes	yes	yes	yes	no	no	yes	yes	no	no	yes
Dev System [42]	yes	yes	yes	yes	yes	no	no	yes	yes	no	no	yes
Elevator [11]	yes	no	yes	no	yes	yes	no	no	no	TO	no	no
Elevator-dashed [11]	yes	no	yes	no	no	no	no	no	no	TO	no	no
Elevator-directed [11]	yes	no	yes	no	no	no	no	no	no	TO	no	no
Filter Collaboration [50]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Four Player Game [36]	yes	yes	yes	no	no	yes	no	yes	no	yes	yes	no
Health System [37]	yes	yes	yes	yes	yes	no	no	yes	yes	no	no	yes
Logistic [41]	yes	yes	yes	yes	yes	yes	no	yes	no	no	no	yes
Sanitary Agency (mod) [44]	yes	yes	yes	yes	yes	yes	no	yes	yes	TO	no	yes
TPM Contract [28]	yes	yes	yes	no	yes	yes	no	yes	no	no	no	no
2-Paxos 2P3A (App F)	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes
Promela I* [18]	yes	no	yes	no	yes	yes	no	yes	yes	yes	yes	yes
Web Services* [18]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Trade System* [18]	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes
Online Stock Broker* [18]	no	no	no	no	no	no	no	yes	no	no	no	yes
FTP* [18]	yes	yes	yes	yes	yes	yes	no	yes	no	no	no	yes
Client-server* [18]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Mars Explosion* [18]	yes	yes	yes	yes	yes	yes	no	yes	no	no	no	yes
Online Computer Sale* [18]	no	yes	no	yes	yes	yes	no	yes	no	no	no	yes
e-Museum* [18]	yes	yes	yes	no	yes	no	no	yes	yes	no	no	yes
Vending Machine* [18]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Bug Report* [18]	yes	yes	yes	no	yes	yes	no	yes	no	no	no	yes
Sanitary Agency* [18]	no	yes	no	yes	yes	yes	no	yes	yes	no	no	yes
SSH* [18]	no	yes	no	yes	yes	yes	no	yes	yes	yes	no	yes
Booking System* [18]	no	yes	no	yes	yes	yes	no	yes	yes	no	no	yes
Hand-crafted Example* [18]	no	yes	no	yes	yes	no	no	yes	yes	no	no	yes

Conclusion

To summarize:

- ▶ introduction of i-RSC and weak k -MC system
- ▶ inclusion in these subclasses is **decidable**
- ▶ translation from session types preserves the semantics