Global Escape in Multiparty Sessions

Sara Capecchi

joint work with Elena Giachino & Nobuko Yoshida

Workshop on Behavioural Types
21 April 2011
Global escape

unexpected condition, computational error
controlled structured interruption requested by some participant
Interactional exceptions (Structured Interactional Exceptions for Session Types. Carbone, Honda, Yoshida. CONCUR’08)
not only local but also coordinated actions among communicating peers: exception affects a collection of parallel processes and an escape needs to move into another dialogue in a concerted manner
Global escape

- unexpected condition, computational error
Global escape

- unexpected condition, computational error
- controlled structured interruption requested by some participant
Global escape

- unexpected condition, computational error
- controlled structured interruption requested by some participant

Interactional exceptions (Structured Interactional Exceptions for Session Types. Carbone, Honda, Yoshida. CONCUR’08)

*not only local but also coordinated actions among communicating peers: exception affects a collection of parallel processes and an escape needs to move into another dialogue in a concerted manner*
Goals & Issues

- Extension of multiparty sessions to flexible exception handling:
  - Asynchronous escape at any desired point of a conversation, including nested exceptions;
- Preserve multiparty session properties:
  - Subject Reduction
  - Communication Safety
  - Session Fidelity
  - Progress
- How to model concurrent exceptions
  - Asynchronous notification to multiple partners
  - Nested exceptions
Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;
Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;
- preserve multiparty session properties:
  
  \textit{Subject Reduction} \hspace{1cm} \textit{Communication Safety} \hspace{1cm} \textit{Session Fidelity} \hspace{1cm} \textit{Progress}
Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;
- preserve multiparty session properties:
  - Subject Reduction
  - Communication Safety
  - Session Fidelity
  - Progress
- how to model
Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;

- preserve multiparty session properties:
  - Subject Reduction
  - Communication Safety
  - Session Fidelity
  - Progress

- how to model
  - concurrent exceptions
Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;
- preserve multiparty session properties:
  - Subject Reduction
  - Communication Safety
  - Session Fidelity
  - Progress
- how to model
  - concurrent exceptions
  - asynchronous notification to multiple partners
Goals & Issues

- extension of multiparty sessions to flexible exception handling: asynchronous escape at any desired point of a conversation, including nested exceptions;

- preserve multiparty session properties:
  - Subject Reduction
  - Communication Safety
  - Session Fidelity
  - Progress

- how to model
  - concurrent exceptions
  - asynchronous notification to multiple partners
  - nested exceptions
Fault tolerance needs error isolation to define exactly which part of the system to recover, and to prevent errors from unlimited propagation. One way to control complexity is to restrict interaction and communication: exception contexts are defined as regions in which the same exceptions are treated in the same way.
Coordinated Actions Model

From *Coordinated Exception handling* - Romanovsky et al.

Fault tolerance needs error isolation to define exactly which part of the system to recover, and to prevent errors from unlimited propagation. One way to control complexity is to *restrict interaction and communication*: exception contexts are defined as regions in which the same exceptions are treated in the same way.

Atomic actions

The activity of a group of components constitutes an atomic action if there are no interactions between that group and the rest of the systems for the duration of the activity.
Coordinated Actions

- Robot
- Robot Sensor
- Press
- Press Sensor

Coordinated actions:
- Robot
- Robot Sensor
- Press
- Press Sensor

Enclosing action: remove plate

- turn robot &
- extend arm
- grab plate from press
Coordinated Actions

\[
\{ (s_1, s_2), [s_1, \gamma_{TR}, \gamma_{HTR}]; [s_1, \gamma_{GP}, \gamma_{HGP}], \gamma_{HRP} \}
\]
Coordinated Actions

\[
\left( (s_1, s_2), [s_1, \gamma_{TR}, \gamma_{HTR}]; [s_1, \gamma_{GP}, \gamma_{HGP}], \gamma_{HRP} \right)
\]

\[
\text{Robot} = \text{try}(s_1, s_2)\{\text{try}(s_1)\{P^R\} \text{ catch } \{Q^R\}\} \text{ catch } \{Q’^R\}
\]

\[
\text{RobotSensor} = \text{try}(s_1, s_2)\{\text{try}(s_1)\{P^{RS}\} \text{ catch } \{Q^{RS}\}; \text{try}(s_1)\{P’_{RS}\} \text{ catch } \{Q’^{RS}\}\} \text{ catch } \{Q’’^{RS}\}
\]

\[
\text{Press} = \text{try}(s_1, s_2)\{\text{try}(s_1)\{P^P\} \text{ catch } \{Q^P\}\} \text{ catch } \{Q’^P\}
\]

\[
\text{PressSensor} = \text{try}(s_1, s_2)\{\text{try}(s_1)\{P^S\} \text{ catch } \{Q^{PS}\}\} \text{ catch } \{Q’^{PS}\}
\]
Syntax and Semantics

\[ P,Q ::= \bar{a}[2..n](\bar{s}).P \quad \text{Multicast Request} \]
\[
\begin{align*}
| & a[p](\bar{s}).P \quad \text{Accept} \\
| & r!(\bar{e}) \quad \text{Output} \\
| & r?(\bar{x}).P \quad \text{Input} \\
| & r<l.P \quad \text{Select} \\
| & r \triangleright \{l_i : P_i\}_{i \in I} \quad \text{Branch} \\
| & \text{try}(\bar{r})\{P\} \text{ catch } \{P\} \quad \text{Try-Catch} \\
| & \text{throw}(\bar{r}) \quad \text{Throw}
\end{align*}
\]
\[
\begin{align*}
| & \text{if } e \text{ then } P \text{ else } P \quad \text{Conditional} \\
| & P | P \quad \text{Parallel} \\
| & P; P \quad \text{Sequencing} \\
| & 0 \quad \text{Inaction} \\
| & (\nu n)P \quad \text{Hiding} \\
| & \text{def } D \text{ in } P \quad \text{Recursion} \\
| & X(\bar{e}\bar{s}) \quad \text{Process call} \\
| & s : L \quad \text{Named queue}
\end{align*}
\]

\[ Σ \vdash \text{try}(\bar{r})\{P\} \text{ catch } \{P\} \rightarrow Σ \vdash \text{try}(\bar{r})\{0\} \text{ catch } \{Q\} \]
\[ Σ, \text{throw}(\bar{r}) \vdash \text{try}(\bar{r})\{P\} \rightarrow Σ, \text{throw}(\bar{r}) \vdash Q \}
\]
### Syntax and Semantics

\[ P, Q ::= \bar{a}[2..n](\bar{s}).P \quad \text{Multicast Request} \]
\[ a[p](\bar{s}).P \quad \text{Accept} \]
\[ r!(\bar{e}) \quad \text{Output} \]
\[ r?(\bar{x}).P \quad \text{Input} \]
\[ r < l.P \quad \text{Select} \]
\[ r \triangleright \{ l_i : P_i \}_{i \in I} \quad \text{Branch} \]
\[ \text{try}(\bar{r})\{P\} \text{ catch } \{Q\} \quad \text{Try-Catch} \]
\[ \text{throw}(\bar{r}) \quad \text{Throw} \]

| if e then P else P | Conditional |
| P | Parallel |
| P; P | Sequencing |
| 0 | Inaction |
| (\nu n)P | Hiding |
| def D in P | Recursion |
| X(\bar{e}\bar{s}) | Process call |
| s : L | Named queue |

#### [Thr]
\[ \Sigma \vdash \text{try}(\bar{r})\{C[\text{throw}(\bar{r})] \mid P\} \text{ catch } \{Q\} \]
\[ \rightarrow \Sigma \uplus \text{throw}(\bar{r}) \vdash \text{try}(\bar{r})\{C \mid P\} \text{ catch } \{Q\} \]

#### [RThr]
\[ \Sigma, \text{throw}(\bar{r}) \vdash \text{try}(\bar{r})\{P\} \text{ catch } \{Q\} \rightarrow \Sigma, \text{throw}(\bar{r}) \vdash Q\{s^{\varphi+1} / s^\varphi\}_{s^\varphi \in \bar{r}} \]
\[ (\text{throw}(\bar{r}') \in \Sigma \text{ implies } \text{try}(\bar{r}') \ldots \notin P, \bar{r}' \subseteq \bar{r}) \]

#### [ZThr]
\[ \Sigma \vdash (\nu \bar{s})(\prod_i \mathcal{E}_i[\text{try}(\bar{r})\{0\} \text{ catch } \{Q_i\}])_{i \in 1..n} \rightarrow \Sigma \vdash (\nu \bar{s})(\prod_i \mathcal{E}_i)_{i \in 1..n} \]
\[ (\text{throw}(\bar{r}) \notin \Sigma) \]
Typing

Partial \( \gamma \) ::= \( p_1 \rightarrow p_2 : k\langle \tilde{S} \rangle \mid p_1 \rightarrow p_2 : k\{l_i : \gamma_i\}_{i \in I} \mid \{\tilde{k}, \gamma, \gamma\} \mid \gamma; \gamma \mid \gamma \parallel \gamma \mid \mu t.\gamma \mid t \)

Global \( G \) ::= \( \gamma; \) end \mid end

Sorts \( S \) ::= bool \mid \ldots \mid \langle G \rangle

Goals:
Typing

Partial \( \gamma ::= p_1 \rightarrow p_2 : k\langle \tilde{S} \rangle | p_1 \rightarrow p_2 : k\{l_i : \gamma_i\}_{i \in I} | \{\tilde{k}, \gamma, \gamma\} | \gamma; \gamma | \gamma \parallel \gamma | \mu t. \gamma | t \)

Global \( G ::= \gamma; \text{end} | \text{end} \)

Sorts \( S ::= \text{bool} | \ldots | \langle G \rangle \)

Goals:

- to check that the enclosed try-catch block is listening on a smaller set of channels: independence of the components w.r.t. exceptions
Typing

Partial \( \gamma \ ::= \ p_1 \to p_2 : k \langle \tilde{S} \rangle | p_1 \to p_2 : k \{ l_i : \gamma_i \}_{i \in I} | \{ \tilde{k}, \gamma, \gamma \} | \gamma \cdot \gamma | \gamma \parallel \gamma | \mu t. \gamma | t \)

Global \( G \ ::= \ \gamma; \ \text{end} | \ \text{end} \)

Sorts \( S \ ::= \ \text{bool} | \ldots | \langle G \rangle \)

Goals:

- to check that the enclosed try-catch block is listening on a smaller set of channels: independence of the components w.r.t. exceptions
- to check that no session request or accept occurs inside a try-catch block
Conclusions

Our extension is:

- consistent: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix.
- safe: linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types.

We ensure these properties using:

- an asynchronous linguistic construct for exceptions signalling multi-level queues.
Conclusions

Our extension is:

- **consistent**: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix
Conclusions

Our extension is:

- **consistent**: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix.
- **safe**: linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types.
Conclusions

Our extension is:

- **consistent**: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix
- **safe**: linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types

We ensure these properties using:
Conclusions

Our extension is:

- **consistent**: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix
- **safe**: linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types

We ensure these properties using:

- an asynchronous linguistic construct for exceptions signalling
Conclusions

Our extension is:

- **consistent**: despite asynchrony and nesting of exceptions, communications in default and exception handling conversations do not mix
- **safe**: linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types

We ensure these properties using:

- an asynchronous linguistic construct for exceptions signalling
- multi-level queues