Capabilities, Session Types and Active Objects

Marco Patrignani \(^1\)  Dave Clarke \(^{1,2}\)

\(^1\)iMinds-DistriNet, Dept. Computer Science, KU Leuven

\(^2\)CS Division, Dept. Information Technology, Uppsala University

January 14, 2014
Outline

1. Overview
2. Calculus
3. Future work
Overview

- global heap
Overview

- global heap
- active objects

Marco Patrignani, Dave Clarke

Capabilities, Session Types and Active Objects 3/11
Overview

- global heap
- active objects
- shared resources

Marco Patrignani, Dave Clarke
Overview

- global heap
- active objects
- shared resources
- concurrent access

Marco Patrignani, Dave Clarke

Capabilities, Session Types and Active Objects
Overview

- global heap
- active objects
- shared resources
- concurrent access

Marco Patrignani, Dave Clarke
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access

Marco Patrignani, Dave Clarke
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities

Marco Patrignani, Dave Clarke

Capabilities, Session Types and Active Objects
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
global heap
active objects
shared resources
concurrent access
race conditions
capabilities regulate resource access
active objects acquire capabilities
and access resource only if in possession of capability
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels

Marco Patrignani, Dave Clarke
Capabilities, Session Types and Active Objects 3/11
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels

Marco Patrignani, Dave Clarke

Capabilities, Session Types and Active Objects
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels

Marco Patrignani, Dave Clarke

Capabilities, Session Types and Active Objects 3/11
Overview

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels

Marco Patrignani, Dave Clarke
Capabilities, Session Types and Active Objects

- global heap
- active objects
- shared resources
- concurrent access
- race conditions
- capabilities regulate resource access
- active objects acquire capabilities
- and access resource only if in possession of capability
- add channels
- pass capabilities through channels
Example (bad)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).
Example (bad)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).

$$SYS = (\nu file : R)(P \mid Q)$$

$$P = W(file) . \emptyset$$

$$Q = W(file) . \emptyset$$
Example (bad)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).

$$SYS = (\nu file : R)(P \mid Q)$$

$$P = W(file).\emptyset$$       $$Q = W(file).\emptyset$$

- race condition!
Example (bad)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).

$$SYS = (νfile : R)(P | Q)$$

$$P = W(file).∅$$
$$Q = W(file).∅$$

- race condition!
  - we do not want any of those
Example (bad)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).

$$SYS = (\nu file : R)(P | Q)$$

$$P = W(file).\emptyset \quad \quad Q = W(file).\emptyset$$

- race condition!
  1. we do not want any of those
  2. we want to reason about concurrent operations on resources
Example (good)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).
Example (good)

Two processes \( P \) and \( Q \) write (\( W \)) to a shared resource (\( file \)).

\[
SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)
\]

\[
P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset)
\]

\[
Q = (y(fvar).y(wrvar).W(fvar).\emptyset)
\]
Example (good)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($\text{file}$).

\[
SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)
\]

\[
P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset) \quad Q = (y(fvar).y(wrvar).W(fvar).\emptyset)
\]

\[
S = !\exists \alpha R. !W(\alpha).\text{end}
\]
Example (good)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).

$$SYS = (\nu file^{wr}: W(file) : R)(\nu xy : S)(P|Q)$$

$$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset) \quad Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$$

$$S = !\exists \alpha R. !W(\alpha).end$$

- well-typed
Example (good)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).

$$SYS = (\forall file^{wr:W(file)} : R)(\forall xy : S)(P | Q)$$

$P = (W(file).x(file).x(wr).\emptyset)$ \hspace{1cm} $Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$

$$S = !\exists \alpha R. !W(\alpha).\text{end}$$

- well-typed
- $W(file)$ is ok, $P$ is typed against $wr$, $Q$ is not
Example (good)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).

$$SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q)$$

$P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset)$ $Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$

$$S = !\exists_\alpha R. !W(\alpha).\text{end}$$

- well-typed
- $W(file)$ is ok, $P$ is typed against $wr$, $Q$ is not
- $\emptyset$ in $P$ is typed without $wr$ (was sent in $x\langle wr\rangle$)
Example (good)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).

$$SYS = (\nu file^{wr} : W(file) : R)(\nu xy : S)(P \mid Q)$$

$$P = (W(file).x\langle file \rangle.x\langle wr \rangle.\emptyset) \quad Q = (y(fvar).y(wrvar).W(fvar).\emptyset)$$

$$S = !\exists \alpha R. !W(\alpha).\text{end}$$

- well-typed
- $W(file)$ is ok, $P$ is typed against $wr$, $Q$ is not
- $\emptyset$ in $P$ is typed without $wr$ (was sent in $x\langle wr \rangle$)
- $y(fvar)$ binds $\alpha$ to $fvar$ in the capability received later in $S$
Example (good)

Two processes $P$ and $Q$ write ($W$) to a shared resource ($file$).

\[ SYS = (\nu \text{file}^{wr:W(file)} : R)(\nu xy : S)(P|Q) \]

\[ P = (W(file).x\langle file\rangle.x\langle wr \rangle.\emptyset) \quad Q = (y(fvar).y(wrvar).W(fvar).\emptyset) \]

\[ S = !\exists \alpha R. !W(\alpha).end \]

- well-typed
- $W(file)$ is ok, $P$ is typed against $wr$, $Q$ is not
- $\emptyset$ in $P$ is typed without $wr$ (was sent in $x\langle wr \rangle$)
- $y(fvar)$ binds $\alpha$ to $fvar$ in the capability received later in $S$
- so $W(fvar)$ in $Q$ succeeds
Reduction

\[ SYS = (\nu \text{file}^{\text{wr}}: W(\text{file}) : R)(\nu \text{x} \text{y} : S)(P \mid Q) \]
\[ P = (W(\text{file}) \cdot x\langle \text{file} \rangle . x\langle \text{wr} \rangle . \emptyset) \]
\[ Q = (y(\text{fvar}) \cdot y(\text{wrvar}) . W(\text{fvar}) . \emptyset) \]

\[ SYS \rightarrow \ldots \xrightarrow{\tau} P \mid Q[\text{file} / \text{fvar}] \xrightarrow{\text{tau}} P \mid Q[\text{wr} / \text{wrvar}] \rightarrow \ldots \rightarrow \emptyset \mid \emptyset \]
Reduction

\[ SYS = (\nu file^{wr:W(file)} : R)(\nu xy : S)(P|Q) \]

\[ P = (W(file).x\langle file\rangle.x\langle wr\rangle.\emptyset) \]

\[ Q = (y(fvar).y(wrvar).W(fvar).\emptyset) \]

\[ SYS \rightarrow \ldots \rightarrow P | Q[file/fvar] \xrightarrow{\tau} P | Q[wr/wrvar] \rightarrow \ldots \rightarrow \emptyset | \emptyset \]

- \( W(file) \) happens in \( P \)
SYS = (ν file \textit{wr} : W(file) : R)(ν xy : S)(P | Q)

P = (W(file). x⟨file⟩. x⟨wr⟩. ∅)

Q = (y(fvar). y(wrvar). W(fvar). ∅)

SYS \rightarrow \ldots \xrightarrow{\tau} P | Q [\textit{file}/fvar] \xrightarrow{\tau_{au}} P | Q[\textit{wr}/wrvar] \rightarrow \ldots \rightarrow ∅ | ∅

- \textit{W(file)} happens in \textit{P}
- exchange of the resource
**Reduction**

\[
SYS = (\nu \text{file}^{wr: W(\text{file}) : R})(\nu x y : S)(P \mid Q)
\]

\[
P = (W(\text{file}).x(\langle \text{file} \rangle . x(\langle wr \rangle . \emptyset))
\]

\[
Q = (y(fvar). y(\langle wrvar \rangle . W(fvar).\emptyset))
\]

\[
SYS \rightarrow \ldots \xrightarrow{\tau} P \mid Q[\text{file}/fvar] \xrightarrow{\text{tau}} P \mid Q[\text{wr}/wrvar] \rightarrow \ldots \rightarrow \emptyset \mid \emptyset
\]

- \(W(\text{file})\) happens in \(P\)
- exchange of the resource and of the \(wr\) capability
SYS = (ν file wr:W(file) : R)(ν xy : S)(P | Q)

P = (W(file).x⟨file⟩.x⟨wr⟩.∅)

Q = (y(fvar).y(wrvar). W(fvar) .∅)

SYS → ... τ→ P | Q[file/fvar] τau→ P | Q[wr/wrvar] → ... → ∅ | ∅

- W(file) happens in P
- exchange of the resource and of the wr capability
- W(file) happens in Q
Idea

- core calculus ($\pi$-calculus inspired)
- session types regulate sharing of capabilities
- capabilities are abstract and linear
- well-typed processes have no race conditions (for some def. of race condition)
Syntax of processes

\[ P ::= \emptyset \mid P \mid P \mid x(y).P \mid x \lhd a.P \mid x\{a_i.P_i\}_{i \in I} \mid (\nu xy : S)P \mid (\nu r^{\overline{c} : \overline{\sigma}} : R)P \mid \varphi(r).P \]
Syntax of processes

\[ P ::= \emptyset \quad | \quad P \mid P \quad | \quad x\langle y \rangle.P \quad | \quad x\triangleleft a.P \quad | \quad x\{a_i.P_i\}_{i \in I} \quad | \quad (\nu xy : S)P \quad | \quad (\nu r^c : \sigma : R)P \quad | \quad \varphi(r).P \]

- standard $\pi$-calculus + sessions (Vasconcelos)
Syntax of processes

\[
P ::= \emptyset | P \parallel P | x\langle y \rangle . P | x \triangleleft a . P | x . \{ a_i . P_i \}_{i \in I} | (\nu x y : S) P | (\nu r^{\overline{c} : \sigma} : R) P | \varphi (r) . P
\]

- standard $\pi$-calculus + sessions (Vasconcelos)
- co-creation of resource $r$ and capabilities $\overline{c}$
- capabilities have type $\sigma$
Syntax of processes

\[ P ::= \emptyset \mid P \parallel P \mid x\langle y\rangle.P \mid x \triangleleft a.P \mid x\{a_i.P_i\}_{i \in I} \mid (\nu xy : S)P \mid (\nu r\overline{c} : \sigma : R)P \mid \varphi(r).P \]

- standard \(\pi\)-calculus + sessions (Vasconcelos)
- co-creation of resource \(r\) and capabilities \(\overline{c}\)
- capabilities have type \(\sigma\)
- (abstract) operation on resource \(r\)
Syntax of processes and types

\[ P ::= ∅ \quad | \quad P \quad | \quad P \quad | \quad x(y).P \quad | \quad x \triangleleft a.P \quad | \quad x.\{a_i.P_i\}_{i \in I} \quad | \quad (νxy : S)P \quad | \quad (νr^{c:\bar{σ}} : R)P \quad | \quad ϕ(r).P \]

\[ S ::= !t.S \quad | \quad &\{a_i : S_i\}_{i \in I} \quad | \quad \?t.S \quad | \quad \oplus \{a_i : S_i\}_{i \in I} \quad | \quad end \quad | \quad end \quad | \quad t ::= S \quad | \quad σ ::= ϕ(r) \quad | \quad σ \quad | \quad ϕ(α) \quad | \quad R \quad | \quad \existsαR \]
Syntax of processes and types

\[ P ::= \emptyset \mid P \mid P \mid x(y).P \mid x \triangle a.P \mid x.a.P \mid \{ a_i . P_i \} \}_{i \in I} \mid (\nu x y : S) P \mid (\nu r c : \sigma : R) P \mid \varphi(r).P \]

\[ S ::= ! t . S \mid \& \{ a_i : S_i \} \}_{i \in I} \mid t ::= S \mid \sigma ::= \varphi(r) \mid ? t . S \mid \oplus \{ a_i : S_i \} \}_{i \in I} \mid \sigma \mid \varphi(\alpha) \mid \text{end} \mid R \mid \exists \alpha R \]

- capability types specify what (abstract) operation can be executed on a resource
Syntax of processes and types

\[ P ::= \emptyset \mid P \mid P \mid x(y).P \mid x \triangleleft a.P \mid x\{a_i.P_i\}_{i \in I} \mid (\nu xy : S)P \mid (\nu r \overline{c:}\overline{\sigma} : R)P \mid \phi(r).P \]

\[ S ::= !t. S \mid \&\{a_i : S_i\}_{i \in I} \mid \oplus \{a_i : S_i\}_{i \in I} \mid \text{end} \mid ?t. S \mid \sigma \mid \varphi(\alpha) \mid R \mid \exists \alpha R \]

- capability types specify what (abstract) operation can be executed on a resource
- existential quantifiers bind resource variables \( \alpha \) in the continuation of the session
Properties

- progress: $\vdash P$ and $P \rightarrow P'$ implies $\vdash P'$
Properties

- progress: ⊢ $P$ and $P \rightarrow P'$ implies ⊢ $P'$
- no errors:

\[ \phi(r) \equiv \phi'(r') \]

If $\phi(r) = \phi'(r')$, the order of execution of $\phi(r)$ and $\phi'(r')$ does not matter.
Properties

- progress: \( \vdash P \) and \( P \rightarrow P' \) implies \( \vdash P' \)
- no errors:
  - no race conditions: \( \vdash P \) implies \( P \not\rightarrow P' \) and \( P' \equiv \phi(r).Q|\phi'(r').Q' \) where \( \phi(r) \neq \phi'(r') \)

Where \( \phi(r) \simeq \phi'(r') \)
  - if \( \phi(r) \cdot \phi'(r') = \phi'(r') \cdot \phi(r) \)
  (i.e. the order of execution of \( \phi(r) \) and \( \phi'(r') \) does not matter)
So far, this is very much WIP

- borrowing capabilities (auto get back after some time)
So far, this is very much WIP

- borrowing capabilities (auto get back after some time)
- proving properties of the type system
So far, this is very much WIP

- borrowing capabilities (auto get back after some time)
- proving properties of the type system
- add locations (model group of resources as ownership)
So far, this is very much WIP

- borrowing capabilities (auto get back after some time)
- proving properties of the type system
- add locations (model group of resources as ownership) and capabilities on locations
So far, this is very much WIP

- borrowing capabilities (auto get back after some time)
- proving properties of the type system
- add locations (model group of resources as ownership) and capabilities on locations
- more?