Modular Session Types for Objects

Simon Gay, Nils Gesbert, António Ravara, Vasco Vasconcelos

University of Glasgow, INRIA Grenoble – Rhône-Alpes, Universidade Nova de Lisboa, Universidade de Lisboa

19th April 2011
Several methods available: external choice

\{\text{hasNext} : S, \text{close} : S'\}

Object branches / Client selects by calling a method

Dependency on a method result: internal choice

\langle \text{OK} : S, \text{ERROR} : S'\rangle

Object selects by returning a label / Client branches
Session Types for Objects

- Several methods available: external choice
  \{hasNext : S, close : S’\}

  Object branches / Client selects by calling a method

- Dependency on a method result: internal choice
  \(<OK : S, ERROR : S’>\)

  Object selects by returning a label / Client branches

\[
\text{session Init} \\
\text{where Init} = \{ \text{open: } \langle \text{OK: Open, ERROR: Init} \rangle \} \\
\text{Open} = \{ \text{hasNext: } \langle \text{TRUE: Read, FALSE: Close} \rangle, \text{close: Init} \} \\
\text{Read} = \{ \text{read: Open, close: Init} \} \\
\text{Close} = \{ \text{close: Init} \}
\]
How it works

- Calling a method advances the session type of the object
How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must **switch** on the result to resolve it
How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must **switch** on the result to resolve it **but the result can be stored and switched on later**
How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must switch on the result to resolve it but the result can be stored and switched on later
- Objects are linear but may be stored in fields of other objects
How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must switch on the result to resolve it but the result can be stored and switched on later
- Objects are linear but may be stored in fields of other objects
- External (abstract) type of an object: session type, $S$
How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must **switch** on the result to resolve it but the result can be stored and switched on later
- Objects are linear but may be stored in fields of other objects
- External (abstract) type of an object: session type, $S$
  
  $S$ contains method signatures
How it works

- Calling a method advances the session type of the object.
- If the continuation is an internal choice, the client must switch on the result to resolve it. **But the result can be stored and switched on later.**
- Objects are linear but may be stored in fields of other objects.
- External (abstract) type of an object: session type, $S$.
  
  $S$ contains method signatures.

- Internal state of an object: type of its fields, $C[F]$. 
How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must switch on the result to resolve it but the result can be stored and switched on later
- Objects are linear but may be stored in fields of other objects
- External (abstract) type of an object: session type, $S$
  
  $S$ contains method signatures

- Internal state of an object: type of its fields, $C[F]$

Judgements:

- Expressions: $\Gamma \ast r \triangleright e : T \triangleleft \Gamma' \ast r'$

  $r = \text{current object}$
How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must `switch` on the result to resolve it but the result can be stored and switched on later
- Objects are linear but may be stored in fields of other objects
- External (abstract) type of an object: session type, $S$
  - $S$ contains method signatures
- Internal state of an object: type of its fields, $C[F]$

Judgements:

- Expressions: $\Gamma \star r \triangleright e : T \triangleleft \Gamma' \star r'$
  - $r = \text{current object}$
- For a method body:
  - $\text{this} : C[F], x : T' \star \text{this} \triangleright e : T \triangleleft \text{this} : C[F'] \star \text{this}$
How it works

- Calling a method advances the session type of the object
- If the continuation is an internal choice, client must switch on the result to resolve it but the result can be stored and switched on later
- Objects are linear but may be stored in fields of other objects
- External (abstract) type of an object: session type, $S$
  - $S$ contains method signatures
- Internal state of an object: type of its fields, $C[F]$

Judgements:

- Expressions: $\Gamma \ast r \triangleright e : T \triangleleft \Gamma' \ast r'$
  - $r =$ current object
- For a method body:
  - $\text{this} : C[F], \text{x} : T' \ast \text{this} \triangleright e : T \triangleleft \text{this} : C[F'] \ast \text{this}$
- Internal/External state compatibility: $F \vdash C : S$
  - Coinductively checks method bodies in order
Subtyping

Coinductively defined on sessions:

- An object with more methods can be safely used in place of an object with less methods
- An object with less internal choice (more deterministic) can be safely used in place of an object with more internal choice
- Covariance on result types and continuation session, contravariance on parameter types
Subtyping

Coinductively defined on sessions:
- An object with more methods can be safely used in place of an object with less methods
- An object with less internal choice (more deterministic) can be safely used in place of an object with more internal choice
- Covariance on result types and continuation session, contravariance on parameter types

Properties:
- if $F' <: F$ and $F \vdash C : S$ then $F' \vdash C : S$
- if $S <: S'$ and $F \vdash C : S$ then $F \vdash C : S'$
If field $f$ has type $\{ T' \ m(T) : \langle l : S_l \rangle_{l \in E}, \ldots \}$
Link types

If field $f$ has type $\{ T' \; m(T) : \langle l : S_l \rangle_{l \in E}, \ldots \}$ then:

- the result of $f.m(x)$ is a label $l$ in $E$
- this result indicates which state $f$ is in
Link types

If field $f$ has type $\{ T' \; m(T) : \langle l : S_i \rangle_{l \in E}, \ldots \}$ then:

- the result of $f.m(x)$ is a label $l$ in $E$
- this result indicates which state $f$ is in
- it has type link $f$
If field $f$ has type $\{ T' \; m(T) : \langle l : S_i \rangle_{i \in E}, \ldots \}$ then:

- the result of $f.m(x)$ is a label $l$ in $E$
- this result indicates which state $f$ is in
- it has type link $f$
- $T'$ is link this
Link types

If field \( f \) has type \( \{ T' \ \text{m}(T) : \langle l : S_l \rangle_{l \in E}, \ldots \} \) then:

- the result of \( f \cdot \text{m}(x) \) is a label \( l \) in \( E \)
- this result indicates which state \( f \) is in
- it has type \text{link} \( f \)
- \( T' \) is \text{link}this

In the body of \( m \), a \textit{variant field typing} is constructed
Selected Typing Rules

(T-Label) \( \Gamma \ast r \triangleright l : \{l\} < \Gamma \ast r \)

(T-New) \( \Gamma \ast r \triangleright \text{new } C() : C.\text{session} < \Gamma \ast r \)

\[
\Gamma \ast r \triangleright e : T'_j < \Gamma' \ast r' \\
\Gamma'(r'.f) = \{ T_i : m_i(T'_i) : S_i \}_{i \in I} \\
j \in I \\
T = \text{link } f \text{ if } T_j = \text{linkthis, } T = T_j \text{ otherwise} \\
\]

\[
\Gamma \ast r \triangleright f.m_j(e) : T < \Gamma'\{r'.f \mapsto S_j\} \ast r' \\
\]

(T-Call)

(T-SwitchLink)

\[
\Gamma \ast r \triangleright e : \text{link } f < \Gamma' \ast r' \\
\Gamma'(r'.f) = \langle l : S_l \rangle_{l \in E'} \\
E' \subseteq E \\
\forall l \in E', \Gamma'\{r'.f \mapsto S_l\} \ast r' \triangleright e_l : T < \Gamma'' \ast r' \\
\]

\[
\Gamma \ast r \triangleright \text{switch } (e) \{ l : e_l \}_{l \in E} : T < \Gamma'' \ast r' \\
\]

(T-VarF)

\[
\Gamma \ast r \triangleright e : E < \Gamma' \ast r' \\
\Gamma'(r') = C[F'] \\
F' \text{ is a record} \\
\]

\[
\Gamma \ast r \triangleright e : \text{linkthis} < \Gamma'\{r' \mapsto C[\langle l : F' \rangle_{l \in E}]\} \ast r' \\
\]

(T-Class)

\[
\text{Null } \vec{f} \vdash C : S \\
\vdash \text{class } C \{ S ; \vec{f} ; \vec{M} \} \\
\]
Properties of the sequential system

- **Subject Reduction**
  - program state = heap, expression, current object: $$(h \ast r; e)$$
  - internal system checks compatibility between $\Gamma$ and $h$

- **Progress**
  - if $$(h \ast r; e)$$ is well-typed then either $e$ is a value or $$(h \ast r; e)$$ reduces

- **Conformance**
  - the sequence of method calls on an object is a trace of the declared session of its class

Implementation: Bica (demo by Zua Caldeira at 4:30)
Translation of channel session types to class session types

\[
\begin{align*}
\llbracket X \rrbracket &= X \\
\llbracket \mu X.\Sigma \rrbracket &= \mu X.\llbracket \Sigma \rrbracket \\
\llbracket ? [T] . \Sigma \rrbracket &= \{T \text{ receive}(\text{Null}) : \llbracket \Sigma \rrbracket\} \\
\llbracket ! [T] . \Sigma \rrbracket &= \{\text{Null send}(T) : \llbracket \Sigma \rrbracket\} \\
\llbracket \& \{l : \Sigma_l\}_{l \in E} \rrbracket &= \{\text{linkthis receive(Null)} : \langle l : \llbracket \Sigma_l \rrbracket \rangle_{l \in E}\} \\
\llbracket \oplus \{l : \Sigma_l\}_{l \in E} \rrbracket &= \{\text{Null send}\{\{l\}\} : \llbracket \Sigma_l \rrbracket\}_{l \in E}\end{align*}
\]
Example: communicating with a file server

File server with channel session type:

FileChannel = &\{OPEN: ?String . \oplus \{OK: CanRead , ERROR: FileChannel\} ,
                   QUIT: End\}
CanRead = &\{READ: \oplus\{EOF: FileChannel , DATA: !String . CanRead\},
               CLOSE: FileChannel\}

Translated (client-side) as:

ClientCh = \{send (\{OPEN\}): \{send (String ): \{receive: \langle OK: CanRead ,
                                               ERROR: ClientCh \rangle \}\},
             send (\{QUIT\}): \{\}\}\}
CanRead = \{send (\{READ\}): \{receive: \langle EOF: ClientCh ,
                                     DATA: \{receive: CanRead\}\rangle \},
            send (\{CLOSE\}): ClientCh\}

Would like to expose interface:

\texttt{session Init}

where \texttt{Init} = \{open: \langle OK: Open , ERROR: Init \rangle\}
\texttt{Open} = \{hasNext: \langle TRUE: Read , FALSE: Close \rangle ,
                close: Init\}
\texttt{Read} = \{read: Open ,
                close: Init\}
\texttt{Close} = \{close: Init\}
Subject Reduction

Communication Safety (as with usual binary session types)
For any class $C$, we define the relation $F \vdash C : S$ between field typings $F$ and session types $S$ as the largest relation such that $F \vdash C : S$ implies:

- If $S \equiv \{ T_i \; m_i(T_i') : S_i \}_{i \in I}$, then $F$ is not a variant and for all $i$ in $I$, there is a definition $m_i(x_i) \{ e_i \}$ in the declaration of class $C$ such that we have $F ; x_i : T_i' \triangleright e_i : T_i \triangleleft F_i ; \emptyset$ with $F_i$ such that $F_i \vdash C : S_i$.

- If $S \equiv \langle l : S_l \rangle_{l \in E}$, then $F = \langle l : F_l \rangle_{l \in E'}$ with $E' \subseteq E$ and for any $l$ in $E'$ we have $F_l \vdash C : S_l$. 
Operational semantics

\[(R\text{-Seq})\quad (h\ast r; \; v; \; e) \rightarrow (h\ast r; \; e)\]

\[(R\text{-Call})\quad \frac{m(x) \{ e \} \in h(r.f).\text{class}}{(h\ast r; \; f.m(v)) \rightarrow (h\ast r.f; \; \text{return } e\{v/x\})}\]

\[(R\text{-Return})\quad (h\ast r.f; \; \text{return } v) \rightarrow (h\ast r; \; v)\]

\[(R\text{-Switch})\quad \frac{l_0 \in E}{(h\ast r; \; \text{switch } (l_0) \{ l : e_l \}_{l \in E}) \rightarrow (h\ast r; \; e_{l_0})}\]

\[(R\text{-Swap})\quad \frac{h(r).f = v}{(h\ast r; \; f \leftrightarrow v') \rightarrow (h\{r.f \leftrightarrow v'\} \ast r; \; v)}\]

\[(R\text{-New})\quad \frac{o \text{ fresh}}{(h\ast r; \; \text{new } C()) \rightarrow (h, \{ o = C[\overrightarrow{f} = \text{null}] \} \ast r; \; o)}\]

\[(R\text{-Context})\quad \frac{(h\ast r; \; e) \rightarrow (h'\ast r'; \; e')}{(h\ast r; \; E[e]) \rightarrow (h'\ast r'; \; E[e'])}\]