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

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

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

```
session Init
where Init  = {open: ⟨OK: Open, ERROR: Init⟩}
      Open  = {hasNext: ⟨TRUE: Read, FALSE: Close⟩, close: Init}
      Read  = {read: Open, close: Init}
      Close = {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, 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: $\Gamma * r \triangleright e : T \triangleleft \Gamma' * r'$ $r =$ 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: $\Gamma * r \triangleright e : T \triangleleft \Gamma' * r'$ $r =$ current object
- For a method body:
  $\text{this} : C[F], x : T' * \text{this} \triangleright e : T \triangleleft \text{this} : C[F'] * \text{this}$

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: $\Gamma * r \triangleright e : T \triangleleft \Gamma' * r'$ r = current object
- For a method body:
  $\mathsf{this} : C[F], x : T' * \mathsf{this} \triangleright e : T \triangleleft \mathsf{this} : C[F'] * \mathsf{this}$
- 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 $F \vdash C : S'$

Link types

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 $\{T'\ m(T) : \langle l : S_l \rangle_{l \in E}, \ldots\}$ then:

- the result of $f.m(x)$ is a label l in E
- this result indicates which state f is in
- it has type link f

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
- it has type link f
- T' is linkthis

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
- it has type link f
- T' is linkthis

In the body of m, a variant field typing is constructed

Selected Typing Rules

$$(\text{T-Label})\quad \Gamma * r \triangleright l : \{l\} \triangleleft \Gamma * r$$

$$(\text{T-New})\quad \Gamma * r \triangleright \text{new } C() : C.\text{session} \triangleleft \Gamma * r$$

$$(\text{T-Call})\ \frac{\Gamma * r \triangleright e : T'_j \triangleleft \Gamma' * r' \qquad \Gamma'(r'.f) = \{T_i\ m_i(T'_i) : S_i\}_{i \in I} \qquad j \in I \qquad T = \text{link } f \text{ if } T_j = \text{linkthis},\ T = T_j \text{ otherwise}}{\Gamma * r \triangleright f.m_j(e) : T \triangleleft \Gamma'\{r'.f \mapsto S_j\} * r'}$$

$$(\text{T-SwitchLink})\ \frac{\Gamma * r \triangleright e : \text{link } f \triangleleft \Gamma' * r' \qquad \Gamma'(r'.f) = \langle l : S_l \rangle_{l \in E'} \qquad E' \subseteq E \qquad \forall l \in E', \Gamma'\{r'.f \mapsto S_l\} * r' \triangleright e_l : T \triangleleft \Gamma'' * r'}{\Gamma * r \triangleright \text{switch } (e)\ \{l : e_l\}_{l \in E} : T \triangleleft \Gamma'' * r'}$$

$$(\text{T-VarF})\ \frac{\Gamma * r \triangleright e : E \triangleleft \Gamma' * r' \qquad \Gamma'(r') = C[F'] \qquad F' \text{ is a record}}{\Gamma * r \triangleright e : \text{linkthis} \triangleleft \Gamma'\{r' \mapsto C[\langle l : F' \rangle_{l \in E}]\} * r'}$$

$$(\text{T-Class})\ \frac{\overrightarrow{\text{Null}}\ \vec{f} \vdash C : S}{\vdash \text{class } C\ \{S; \vec{f}; \vec{M}\}}$$

Properties of the sequential system

- Subject Reduction
  - program state = heap, expression, current object: $(h * r;\ e)$
  - internal system checks compatibility between Γ and h
- Progress
  - if $(h * r;\ e)$ is well-typed then either e is a value or $(h * r;\ e)$ 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)

Channels

Translation of channel session types to class session types

$$
\begin{aligned}
[\![X]\!] &= X \\
[\![\mu X.\Sigma]\!] &= \mu X.[\![\Sigma]\!] \\
[\![?\,[T]\,.\,\Sigma]\!] &= \{T\ \mathsf{receive}(\mathsf{Null}) : [\![\Sigma]\!]\} \\
[\![!\,[T]\,.\,\Sigma]\!] &= \{\mathsf{Null}\ \mathsf{send}(T) : [\![\Sigma]\!]\} \\
[\![\&\,\{l : \Sigma_l\}_{l\in E}]\!] &= \{\mathsf{linkthis}\ \mathsf{receive}(\mathsf{Null}) : \langle l : [\![\Sigma_l]\!]\rangle_{l\in E}\} \\
[\![\oplus\,\{l : \Sigma_l\}_{l\in E}]\!] &= \{\mathsf{Null}\ \mathsf{send}(\{l\}) : [\![\Sigma_l]\!]\}_{l\in E}
\end{aligned}
$$

Example: communicating with a file server

File server with channel session type:

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

Translated (client-side) as:

```
  ClientCh = {send({OPEN}):{send(String):{receive:⟨OK:CanRead,
                                                   ERROR:ClientCh⟩}},
              send({QUIT}): {}}
  CanRead = {send({READ}): {receive: ⟨EOF: ClientCh,
                                      DATA: {receive: CanRead}⟩},
             send({CLOSE}): ClientCh}
```

Would like to expose interface:

```
session Init
where Init  = {open: ⟨OK: Open, ERROR: Init⟩}
      Open  = {hasNext: ⟨TRUE: Read, FALSE: Close⟩, close: Init}
      Read  = {read: Open, close: Init}
      Close = {close: Init}
```

Results

Subject Reduction

Communication Safety (as with usual binary session types)

$F \vdash C : S$

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

$$\text{(R-Seq)} \quad (h * r;\ v; e) \longrightarrow (h * r;\ e)$$

$$\text{(R-Call)} \ \frac{m(x)\ \{e\} \in h(r.f).\text{class}}{(h * r;\ f.m(v)) \longrightarrow (h * r.f;\ \text{return } e\{^{v}/_{x}\})}$$

$$\text{(R-Return)} \quad (h * r.f;\ \text{return } v) \longrightarrow (h * r;\ v)$$

$$\text{(R-Switch)} \ \frac{l_0 \in E}{(h * r;\ \text{switch } (l_0)\ \{l : e_l\}_{l \in E}) \longrightarrow (h * r;\ e_{l_0})}$$

$$\text{(R-Swap)} \ \frac{h(r).f = v}{(h * r;\ f \leftrightarrow v') \longrightarrow (h\{r.f \mapsto v'\} * r;\ v)}$$

$$\text{(R-New)} \ \frac{o \text{ fresh} \qquad C.\text{fields} = \vec{f}}{(h * r;\ \text{new } C()) \longrightarrow (h, \{o = C[\vec{f} = \overrightarrow{\text{null}}]\} * r;\ o)}$$

$$\text{(R-Context)} \ \frac{(h * r;\ e) \longrightarrow (h' * r';\ e')}{(h * r;\ \mathcal{E}[e]) \longrightarrow (h' * r';\ \mathcal{E}[e'])}$$