PRACTICAL SESSION TYPES

Christophe De Troyer
Christophe Scholliers
Wolfgang De Meuter
SESSION TYPES

JABBER
POP3
SMTP
## State of the Art

<table>
<thead>
<tr>
<th>Calculi</th>
<th>Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Calculi</td>
<td>Haskell, Scala, OCaml, Links, Sill, C, Erlang, Go, Rust, Java</td>
</tr>
<tr>
<td>π-calculus</td>
<td></td>
</tr>
<tr>
<td>Object-Oriented</td>
<td></td>
</tr>
</tbody>
</table>
Running Example

+ Session Types in a Nutshell
Hand over ticket

Hand over passport

Receive boarding pass

Boarding denied

Hand over luggage
Hand over ticket

Hand over passport

Hand over luggage

Receive boarding pass

Boarding denied

OK

NOK

!T = Send type T
?T = Receive type T
. = Sequence

!Ticket.!Passport.&<OK : ?BoardingPass.!Luggage.End,
NOK: ?String.End

>
Duality

!Ticket.!Passport.&<OK : ?BoardingPass.!Luggage.End,
NOK: ?String.End

?Ticket.?Passport.⊕<OK : !BoardingPass.?Luggage.End,
NOK: !String.End
Modularity is difficult

Types

Logic

Refactor

Type information explodes
Improving modularity at the type level in session type programs

- Polymorphism for full session types
- Pattern Matching
- Session types in System F
Session Types in Practice
A FUNCTION WORKING ON TYPES

Session Typed Function

cHECKIN :: [\beta:S] Chan \beta \rightarrow [\beta:S'] Bool
A SAMPLE PROGRAM


checkin :: [ß:Checkin] Chan β -> [ß:End] Chan β
checkin airport = let ticket = grabTicket
passport = grabPassport
luggage = grabFromTrunk
in
send ticket airport ;
send passport airport ;
case airport
  OK = let boardingpass = receive airport
       in
       send luggage airport ;
close airport
NOK = let err = receive airport
     in
     close airport ;
error (format “Go back home: %s” err)
A SAMPLE PROGRAM

```plaintext
session Checkin = !Ticket.!Passport.&<OK : ?BoardingPass.!Luggage.End,
                 NOK: End

checkin :: [ß:Checkin] Chan ß -> [ß:End] Chan ß
checkin airport = let ticket   = grabTicket
                 passport = grabPassport
                 luggage  = grabFromTrunk
               in
               send ticket airport ;
               send passport airport ;
               case airport
                 OK  = let boardingpass = receive airport
                        in
                        send luggage airport ;
                        close airport
                 NOK = let err = receive airport
                        in
                        close airport ;
                        error (format "Go back home: %s" err)
```
A sample program


checkin :: [ß:Checkin] Chan ß -> [ß:End] Chan ß

checkin airport = let ticket = grabTicket
                  passport = grabPassport
                  luggage = grabFromTrunk

                  in
                  send ticket airport ;
                  send passport airport ;

                  case airport
                  OK  = let boardingpass = receive airport
                        in
                        send luggage airport ;
                        close airport
                  NOK = let err = receive airport
                        in
                        close airport ;
                        error (format “Go back home: %s” err)
Modularity is difficult

Types

Logic

Refactor

Type information explodes
#1: Polymorphism for session types
Polymorphism

sendBack :: ForAll T . [β: ?T.!T.End] Chan β -> [β:End]()
Apply f to the received value and send it back

```
sendBack :: ForAll T,U . [ß: ?T.!U.End] Chan ß -> (T -> U) -> [ß:End]()
sendBack chn f = let incoming = receive chn
                 outgoing = f incoming
                 in
                  send outgoing chn
```

??T

!U
POLYMORPHISM

```
session Checkin = ⊕<NORMAL: !Ticket.!Passport,
          FAST : !Barcode >.&<OK : ?BoardingPass.!Luggage.End,
          NOK: End>

customsHandbag conveyorbelt = let handbag = receive conveyorbelt in
  if (containsLiquids? handbag)
  then select NOK conveyorbelt
  send “Not allowed!”
  else send handbag conveyorbelt

customsLuggage conveyorbelt = let luggage = receive conveyorbelt in
  if (containsDrugs? handbag)
  then select NOK conveyorbelt
  send “Not allowed!”
  else send handbag conveyorbelt
```
Polymorphism for Session Types

```session
Checkin = ⊗<NORM: !Ticket.!Passport,
         FAST : !Barcode >.&<OK : ?BoardingPass.!Luggage.End,
         NOK: End >
```

```checkWith
```

```checkWith conveyorbelt f = let t = receive conveyorbelt
    in
    if (f t)
       then select ERR conveyorbelt
           send “Not allowed!”
       else send t conveyorbelt```

```customsHandbag
customsHandbag conveyorbelt = checkWith conveyorbelt containsLiquids?
```

```customsLuggage
customsLuggage conveyorbelt = checkWith conveyorbelt containsDrugs?
```
#2: Pattern Matching
Enforce Structural constraints

“As long as the session type wants me to send an Int”

\[
f :: \left[\beta : S. \texttt{!Int}.U\right] \text{Chan } \beta \rightarrow \left[\beta : U\right] ()
\]

“As long as the channel offers these two choices”

\[
h :: \left[\beta : \&<\texttt{CH1}: S, \texttt{CH2}: T>.U\right] \text{Chan } \beta \rightarrow \left[\beta : U\right] ()
\]
A SAMPLE PROGRAM

sendTicket :: [β:Checkin] Chan β
->

sendTicket airport = let ticket = grabTicket in send ticket airport

->

sendPassport airport = let passport = grabPassport in send passport airport

getBoardingPass :: [β:?BoardingPass.!Luggage.End] Chan β
->
  [β:!Luggage.End] Chan β

getBoardingPass airport = return (receive airport)
selectFastTrack :: \([eta: +<\text{FAST}: S, \text{NORMAL: } S'>.S'']\] Chan \beta \rightarrow \[\beta:S.S'']\] Chan \beta
selectFastTrack airport = select FAST airport

sendTicket :: \[\beta:!\text{Ticket}.S\] Chan \beta \rightarrow \[\beta:S\] Chan \beta
sendTicket airport = let ticket = grabTicket in send ticket airport

sendPassport :: \[\beta:!\text{Passport}.S\] Chan \beta \rightarrow \[\beta:S\] Chan \beta
sendPassport airport = let passport = grabPassport in send passport airport

getBoardingPass :: \[\beta:?\text{BoardingPass}.S\] Chan \beta \rightarrow \[\beta:S\] BoardingPass
getBoardingPass airport = return (receive airport)
Thank you!

- Help session types transform with program
- Reduce type overhead
- Proof of Concept implementation
  - Polymorphism for full session types
  - Pattern matching (Structural Constraints)
- System F with session types
- All images taken from pixabay.com