Gradual Session Types

Albert-Ludwigs-Universität Freiburg

Atsushi Igarashi\textsuperscript{1} \quad Peter Thiemann\textsuperscript{2} \quad Vasco Vasconcelos\textsuperscript{3}

\textsuperscript{1}Kyoto University, \textsuperscript{2}University of Freiburg, \textsuperscript{3}University of Lisbon

31 Jan 2017
Session Types are Good for You

- communication on typed channels
- bidirectional and heterogeneous
- type safety and communication safety
- (scalable to multi-session)
- (deadlock freedom)
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[ cccp = \! CCY. \! \text{Double.} \ cccp\text{-loop} \]
\[ cccp\text{-loop} = \& \{ \text{MORE: } ? CCY. ? \text{Double.} \ cccp\text{-loop,} \]
\[ \text{EXIT: } \text{end} \} \]

Send a currency abbreviation \{element of enumeration type\}
Send an amount
\textbf{while} received MORE \textbf{do}
  Receive a currency abbreviation
  Receive an amount
received EXIT
Close the channel
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[
cccp = !\text{CCY.}!\text{Double. } cccp\text{-loop}
\]

\[
cccp\text{-loop} = \&\{ \text{MORE: } ?\text{CCY.} ?\text{Double. } cccp\text{-loop}, \\
\text{EXIT: } \text{end}\}
\]

Send a currency abbreviation \{element of enumeration type\}
Send an amount
**while** received MORE **do**
   Receive a currency abbreviation
   Receive an amount
received EXIT
Close the channel
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[
\text{cccp} = \! \text{CCY.} \! \text{Double. cccp-loop}
\]

\[
\text{cccp-loop} = \&\{ \text{MORE : } ? \text{CCY.} ? \text{Double. cccp-loop,}
\]

\[
\text{EXIT : end}\}
\]

Send a currency abbreviation \{element of enumeration type\}
Send an amount

while received MORE do
  Receive a currency abbreviation
  Receive an amount
received EXIT
Close the channel
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[
cccp = \downarrow \text{CCY.} \uparrow \text{Double. } cccp\text{-loop}
\]
\[
cccp\text{-loop} = \&\{ \text{MORE: } ? \text{CCY.} ? \text{Double. } cccp\text{-loop,}
\text{EXIT: } end\}
\]

Send a currency abbreviation \{element of enumeration type\}
Send an amount
\textbf{while} received \\textbf{MORE} \textbf{do}
  \hspace{1em} Receive a currency abbreviation
  \hspace{1em} Receive an amount
received EXIT
Close the channel
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[ cccp = !CCY. !Double. cccp-loop \]
\[ cccp-loop = &\{ \text{MORE} : \ ?CCY. ?Double. cccp-loop, \]
\[ \quad \text{EXIT} : \ end \}\]

Send a currency abbreviation \{element of enumeration type\}
Send an amount

\textbf{while received MORE do}
  Receive a currency abbreviation
  Receive an amount
received EXIT
Close the channel
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[
cccp = !\text{CCY.}!\text{Double.} \ cccp\text{-}loop
\]
\[
cccp\text{-}loop = &\{ \ \text{MORE} : \ ?\text{CCY.} \ ?\text{Double.} \ cccp\text{-}loop, \ \text{EXIT} : \ \text{end}\}\]

Send a currency abbreviation \{element of enumeration type\}
Send an amount
\textbf{while} received \textbf{MORE} \textbf{do}
  \hspace{1cm} Receive a currency abbreviation
  \hspace{1cm} Receive an amount
received EXIT
Close the channel
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[
cccp = !\text{CCY.}!\text{Double. } cccp\text{-loop}
\]

\[
cccp\text{-loop} = &\{ \text{MORE: } ?\text{CCY.} ?\text{Double. } cccp\text{-loop,}
\]

\[
\text{EXIT: } \text{end}\}
\]

Send a currency abbreviation \{element of enumeration type\}
Send an amount
\textbf{while} received \textbf{MORE} \textbf{do}

Receive a currency abbreviation
\textbf{Receive an amount}

received \textbf{EXIT}
Close the channel
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[
ccc = \text{!CCY.} \text{!Double. ccpp-loop}
\]

\[
ccpp\text{-loop} = &\{ \text{MORE: ?CCY.} \text{?Double. ccpp-loop,} \\
\text{EXIT: end}\}
\]

Send a currency abbreviation \{element of enumeration type\}
Send an amount
\textbf{while} received \textbf{MORE} \textbf{do}
  \hspace{1em} Receive a currency abbreviation
  \hspace{1em} Receive an amount
received \textbf{EXIT}
Close the channel
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[ cccp = \! \text{CCY.} \! \text{Double.} \ cccp\text{-}loop \]
\[ cccp\text{-}loop = \&\{ \text{MORE : } ? \text{CCY.} ? \text{Double.} \ cccp\text{-}loop, \]
\[ \text{EXIT : } \text{end} \}\n
Send a currency abbreviation \{element of enumeration type\}
Send an amount
\textbf{while} received \text{MORE} \textbf{do}
\hspace{1em} Receive a currency abbreviation
\hspace{1em} Receive an amount
\hspace{1em} received \text{EXIT}
Close the channel
Example: CurrenCy Conversion Protocol
Client Perspective, fully session typed

\[
\begin{align*}
\text{ccc} & = \text{!CCY. !Double. cccp-loop} \\
\text{cccp-loop} & = \&\{ \text{MORE : ?CCY. ?Double. cccp-loop,} \\
& \quad \text{EXIT : end} \}
\end{align*}
\]

Send a currency abbreviation \{element of enumeration type\}
Send an amount
\textbf{while} received \textbf{MORE} \textbf{do}
\hspace{1em} Receive a currency abbreviation
\hspace{1em} Receive an amount
\hspace{1em} received \textbf{EXIT}
\hspace{1em} \textbf{Close the channel}
Switch of perspective

Let’s look at a SERVER
A CCCP Server

Switch of perspective

Let’s look at a SERVER

Suppose

- the first part (before the loop) is typed, but
- the loop is written in an untyped language.
Switch of perspective

Let’s look at a SERVER

Suppose

- the first part (before the loop) is typed, but
- the loop is written in an untyped language.

Gradual typing

provides the protective interface via casts
Excursion: Gradual Typing

Simply-Typed Lambda Calculus

\[ T, U ::= B \mid T \to U \mid T \times U \]
Excursion: Gradual Typing

Gradual Lambda Calculus

\[ T, U ::= B \mid T \rightarrow U \mid T \times U \mid ? \quad \text{any simple type} \]

- Extended with type `dynamic`?
Excursion: Gradual Typing

Simply-Typed Lambda Calculus with Sessions (Mini-GV)

\[ T, U ::= S \mid B \mid T \rightarrow U \mid T \times U \]
\[ R, S ::= \text{end} \mid !T.S \mid ?T.S \]

- Extended with session types
Excursion: Gradual Typing

Gradual Lambda Calculus with Sessions (Mini-GV)

\[ T, U ::= S \mid B \mid T \rightarrow U \mid T \times U \mid ? \quad \text{any simple type} \]

\[ R, S ::= \text{end} \mid !T.S \mid ?T.S \mid ?? \quad \text{any session type} \]

- Extended with type dynamic ? and dynamic session ?
- Extended with session types
Excursion: Gradual Typing

Gradual Lambda Calculus with Sessions (Mini-GV)

\[
T, U ::= S \mid B \mid T \rightarrow U \mid T \times U \mid \text{?} \quad \text{any simple type}
\]
\[
R, S ::= \text{end} \mid !T.S \mid ?T.S \mid \text{?} \quad \text{any session type}
\]

- Extended with type \text{dynamic ?} and \text{dynamic session ?}
- Extended with \text{session types}
- Mediated by explicit casts \( M : T \Rightarrow U \)
Excursion: Gradual Typing

Gradual Lambda Calculus with Sessions (Mini-GV)

\[ T, U ::= S \mid B \mid T \to U \mid T \times U \mid ? \]

\[ R, S ::= \text{end} \mid !T.S \mid ?T.S \mid ? \]

- Extended with type **dynamic ?** and dynamic session ?
- Extended with **session types**
- Mediated by explicit casts \( M : T \Rightarrow U \)
- Complication: session types are linear
# typed part

def cccp_server (c: CCCP):
    (ccy, c) = receive (c)
    (amt, c) = receive (c)
    amt = toEUR(ccy, amt)  # normalize to EUR
    exchange_rates (CAST(amt : Double => ?),
                   CAST(c : CCCP-LOOP => ?))

# untyped part

def exchange_rates(amt, c):
    for cy, r in available_rates():
        sendDyn (True, c)
        sendDyn (cy, c)
        sendDyn (r * amt, c)
    else:
        sendDyn (False, c)
def exchange_rates(amt, c):
    for cy, r in available_rates():
        sendDyn(True, c) # MORE
        sendDyn(cy, c) # cy
        sendDyn(r * amt, c) # c
    else:
        sendDyn(False, c) # EXIT
def exchange_rates(amt, c):
    for cy, r in available_rates():
        sendDyn(True, c)  # MORE
        sendDyn(cy, c)    # cy
        sendDyn(r * amt, c)  # c
    else:
        sendDyn(False, c)  # EXIT
def exchange_rates(amt, c):
    for cy, r in available_rates():
        sendDyn(True, c)  # MORE
        sendDyn(cy, c)    # cy
        sendDyn(r * amt, c)  # c
    else:
        sendDyn(False, c)  # EXIT
def exchange_rates(amt, c):
    for cy, r in available_rates():
        sendDyn(True, c)  # MORE
        sendDyn(cy, c)    # cy
        sendDyn(r * amt, c)  # c
        sendDyn(False, c)  # EXIT
    else:
        sendDyn(False, c)  # EXIT

Encoding of MORE = 1?
Number between 1 and N?
Uncoordinated use of channel
def exchange_rates(amt, c):
    for cy, r in available_rates():
        sendDyn(True, c) # MORE
        sendDyn(cy, c) # cy
        sendDyn(r * amt, c) # c
    else:
        sendDyn(False, c) # EXIT

    Encoding of MORE = 1?
    Number between 1 and N?
    Uncoordinated use of channel
    Final send could be missing altogether
Casting

**Fundamental Theorem of Software Engineering**

All problems in computer science can be solved by another level of indirection.

David Wheeler/Butler Lampson
Casting

Fundamental Theorem of Software Engineering

All problems in computer science can be solved by another level of indirection.

David Wheeler/Butler Lampson

Casting linear values

- Casting a value of linear type to \( ? \) introduces a wrapper.
- The wrapper is unrestricted.
- Casting the wrapped linear value unwraps it and locks the wrapper.
- Casting the locked wrapper via an alias raises blame.
- Releasing the linear value restores it into the wrapping and releases the lock.
Cast to Dynamic

Cast to DYN

:S => (?)

:S

:(?)
Cast from Dynamic

:(?) => R

:S => R

Cast from DYN

:S

locked

:S => R

Igarashi, Thiemann, Vasconcelos
Gradual Session Types
31 Jan 2017
10 / 19
Cast from Dynamic/Blame

Cast from DYN

(?:) => R

locked

BLAME
Release to Dynamic

```
(S => (?)
:S
: (?)

:locked

(S
```

Release to DYN
Example code: sendDyn / receiveDyn

sendDyn : ? → ? → ?
sendDyn = \( \lambda x : ?. \lambda c : ?. \text{send} \; x \; (c : ? \Rightarrow !?.(\_)) : ? \Rightarrow ? \)

receiveDyn : ? → ?
receiveDyn = \( \lambda c : ?. \text{fst}(\text{receive} \; (c : ? \Rightarrow ??.(\_))) \)
Linear pairs and functions

- needed for handling sessions
- cast to $?$ treated in the same way
Linear pairs and functions

- needed for handling sessions
- cast to \( ? \) treated in the same way

Subtyping

- unrestricted \( \sqsubseteq \) linear
- consistent subtyping (for admissible casts)
Linear pairs and functions

- needed for handling sessions
- cast to ? treated in the same way

Subtyping

- unrestricted \( \sqsubseteq \) linear
- consistent subtyping (for admissible casts)

Done?

No!
Cast from Dynamic to Dynamic

\[
\text{Cast from DYN:} \quad :(?) \Rightarrow R
\]

\[
\text{locked:} \quad :(?)
\]

\[
\text{Cast from DYN:} \quad :(?) \Rightarrow R
\]
Cast from Dynamic to Dynamic

Igarashi, Thiemann, Vasconcelos
Gradual Session Types
31 Jan 2017
Cast from Dynamic to Dynamic — Solution

Cast from DYN:
: (?) => S
: (?) => R
: (?) locked
: ~S => R
locked
: S => S

Igarashi, Thiemann, Vasconcelos
Gradual Session Types
31 Jan 2017
Conclusions

- Gradual session types
- Type soundness, blame preservation
- Fulfill (most of) refined criteria for gradual typing
  - Embedding of typed code
  - Embedding of untyped code
  - Gradual guarantee: n/a