Uniqueness Typing for Resource Management in Message Passing Concurrency

Adrian Francalanza, Edsko de Vries$^1$ and Matthew Hennessy$^1$

First International Workshop on Linearity
September 12, 2009

$^1$The financial support of SFI is gratefully acknowledged.
## Motivation

### Servers

\[
\begin{align*}
\text{TIME}_\text{SRV} & \triangleq \text{rec } X.\text{getTime?}x.x!\langle\text{time}\rangle.X \\
\text{DATE}_\text{SRV} & \triangleq \text{rec } X.\text{getDate?}x.x!\langle\text{date}\rangle.X
\end{align*}
\]

### Client

\[
\begin{align*}
\text{CLIENT}_0 & \triangleq (\nu \text{ret}_1) \text{ getTime!} \langle \text{ret}_1 \rangle. \\
& \quad \text{ret}_1?y. \\
& \quad (\nu \text{ret}_2) \text{ getDate!} \langle \text{ret}_2 \rangle. \\
& \quad \text{ret}_2?z. \\
& \quad P
\end{align*}
\]
Motivation

- timeSrv
- dateSrv
- client
- getTime
- getDate
Motivation

\[ \text{client} \rightarrow \text{getTime} \rightarrow \text{ret}_1 \rightarrow \text{getDate} \rightarrow \text{dateSrv} \]
\[ \text{client} \rightarrow \text{timeSrv} \rightarrow \text{dateSrv} \]
Motivation

Adrian Francalanza, Edsko de Vries and Matthew Hennessy
Uniqueness Typing for MPC
Motivation

Reusing $ret_1$

\[
\text{CLIENT}_1 \triangleq (\nu ret_1) \quad \begin{align*}
&\text{getTime!} \langle ret_1 \rangle. \\
&ret_1?y. \\
&\text{getDate!} \langle ret_1 \rangle. \\
&ret_1?z. \\
&P
\end{align*}
\]
Motivation

Uniqueness Typing for MPC
Motivation

Adrian Francalanza, Edsko de Vries and Matthew Hennessy

Uniqueness Typing for MPC
Motivation

Adrian Francalanza, Edsko de Vries and Matthew Hennessy

Uniqueness Typing for MPC
Motivation

Explicit allocation and deallocation

\[
\text{CLIENT}_2 \triangleq \text{alloc}(x). \\
get\,Time!\langle x \rangle. \\
x?y. \\
get\,Date!\langle x \rangle. \\
x?z. \\
\text{free} \ x. \\
P
\]
Runtime errors

Faulty server

\[ \text{TIME}_{\text{SRV}} \triangleq \text{rec } X. \text{getTime}?x.x!\langle \text{time}\rangle.x!\langle \text{time}\rangle.X \]

Client

\[ \text{CLIENT}_0 \triangleq (\nu \text{ret}_1) \text{ getTime}!\langle \text{ret}_1\rangle. \]
\[ \quad \text{ret}_1?y. \]
\[ \quad (\nu \text{ret}_2) \text{ getDate}!\langle \text{ret}_2\rangle. \]
\[ \quad \text{ret}_2?z. \]
\[ \quad P \]
Faulty server

\[
\text{TIME} \text{Srv} \triangleq \text{rec } X.\text{getTime?}x.x!\langle \text{time} \rangle .x!\langle \text{time} \rangle .X
\]

Reusing \( ret_1 \)

\[
\text{CLIENT}_1 \triangleq (\nu \text{ret}_1) \quad \text{getTime!}\langle \text{ret}_1 \rangle .
\text{ret}_1?y.
\text{ret}_1?z.
P
\]
Explicit allocation and deallocation

\[ \text{CLIENT}_2 \triangleq \text{alloc}(x). \]
\[ \text{getTime!(}x)\). \]
\[ x?y. \]
\[ \text{getDate!(}x)\). \]
\[ x?z. \]
\[ \text{free } x. \]
\[ P \]
Explicit allocation and deallocation

\[ CLIENT_2 \triangleq \begin{align*}
& \text{alloc}(x). \\
& \text{getTime}!\langle x \rangle. \\
& x?y. \\
& \text{getDate}!\langle x \rangle. \\
& x?z. \\
& \text{free } x. \\
& P
\end{align*} \]
Purpose of this paper

- Develop a semantics for the $\pi$-calculus with explicit allocation and deallocation of channels
- Define what we mean by a runtime error (type mismatch and communication on deallocated channels)
- Develop a type system for the language which rejects programs which may exhibit runtime errors
- Prove that well-typed programs have no runtime errors
Type language

\[ T ::= [T]^a \quad \text{(channel type)} \]

\[
\begin{array}{c|c}
 a & \omega \quad \text{(unrestricted)} \\
 & 1 \quad \text{(affine)} \\
 & (\bullet, i) \quad \text{(unique after } i \text{ steps, } i \in \mathbb{N}) \\
\end{array}
\]
Unrestricted channels

Initial situation

Channel $c$ is unrestricted ($\omega$) and shared by many processes
Unrestricted channels

C sends c to A

\(a!c\)
Unrestricted channels

\( A \) sends \( c \) to \( B \)

\( b_2!c \)
Linearity

Initial situation

Channel $c$ is unrestricted ($\omega$) and shared by many processes
Linearity

C sends c to A

\( a!c \)
A sends $c$ to $B$

$b_2!c.P$ ($c \notin \text{fv } P$)
Linearity

C sends c to both A and B

\( a!c \parallel b_1!c \)
Uniqueness

Initial situation

C has *unique* (●) access to channel \( c \)
Uniqueness

C sends $c$ to $A$

$a!c$
Uniqueness

**A sends c to B**

\[ b_2!c.P \quad (c \notin \text{fv } P) \]
**Uniqueness**

C sends c to both A and B

\[ a!c \parallel b_1!c \]
C communicates with B on c

\[ c?x \parallel c!x \]
C communicates with A on c

c?x || c!x
C ignores uniqueness and sends $c$ to lots of processes

$$d!c \parallel e!c \ldots$$
No uniqueness propagation

Initial situation

$A$ is connected to $B_1, B_2, \ldots$ through a shared channel $b$
No uniqueness propagation

A sends the unique channel $c$

Only one process will receive it
Type splitting

Contraction rule

\[
\Gamma, u : T_1, u : T_2 \vdash P \quad T = T_1 \circ T_2 \quad \text{TCON}
\]

\[
\Gamma, u : T \vdash P
\]
Type splitting

Splitting unrestricted channels

\[ [T] \omega = [T] \omega \circ [T] \omega \]

PUNR

Adrian Francalanza, Edsko de Vries and Matthew Hennessy
Uniqueness Typing for MPC
Type splitting

Splitting unique channels

\[
[T]^{\bullet, i} = [T]^1 \circ [T]^{\bullet, i+1}
\]

\[
\text{PUNQ}
\]

\[
\begin{align*}
\text{c:} &\ (\bullet, 1) \\
\Rightarrow \\
\text{c:} &\ (\bullet, 2)
\end{align*}
\]
Subtyping rule

\[
\frac{\Gamma, u : T_2 \vdash P \quad T_1 \prec_s T_2}{\Gamma, u : T_1 \vdash P} \quad \text{TSUB}
\]
Subtyping

From unrestricted to linear

\[
\begin{align*}
\omega & \xrightarrow{ss} 1 \\
\Rightarrow & \\
\frac{c:\omega \Rightarrow \cdots \Rightarrow c:1}{S\text{AFF}}
\end{align*}
\]

Adrian Francalanza, Edsko de Vries and Matthew Hennessy
Uniqueness Typing for MPC
Subtyping

From unique to unrestricted

\[(\bullet, i) \prec_s \omega\]

\[\text{SUNQ}\]
Subtyping lattice

\[
\begin{array}{c}
\bullet \\
\downarrow \\
(\bullet, 1) \\
\downarrow \\
(\bullet, 2) \\
\downarrow \\
\vdots \\
\downarrow \\
\omega \\
\downarrow \\
1
\end{array}
\]
Usefulness of uniqueness

Strong update

$$\Gamma, u: [T_2] \triangleright P \quad \frac{\Gamma, u: [T_1] \triangleright P}{\text{TREV}}$$

Type of \textit{getTime}

\textit{getTime} : $$[[\text{Time}]^1]_\omega$$
Usefulness of uniqueness

Deallocation

\[ \Gamma \vdash P \]
\[ \Gamma, u : [T]^\bullet \vdash \text{free } u.P \]

Type of getDate

\[ \text{getDate} : [[\text{Date}]^1]^\omega \]
Soundness proof

Theorem (Type safety)

If $\Gamma \vdash \sigma \triangleright P$ then $P \not\rightarrow^{err}$.

Theorem (Subject reduction)

If $\Gamma \vdash \sigma \triangleright P$ and $\sigma \triangleright P \rightarrow^{\sigma'} \triangleright P'$ then there exists a environment $\Gamma'$ such that $\Gamma' \vdash \sigma' \triangleright P'$. 
Conclusions

Main contributions

- Uniqueness allows to safely support strong update and deallocation in languages based on MPC.
- We adapted uniqueness to concurrency by taking advantage of the duality between affinity and uniqueness to allow uniqueness to be temporarily violated.

Future work

- Reasoning about well-typed processes
- Higher-order $\pi$-calculus
- Relation to separation-logic semantics of O’Hearn and Hoare?