Uniqueness Typing for Resource Management in Message Passing Concurrency

Edsko de Vries\textsuperscript{1}, Adrian Francalanza and Matthew Hennessy\textsuperscript{1}

Cambridge Concurrency Workshop
July 2010

\textsuperscript{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

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

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

Reusing \( ret_1 \)

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

timeSrv

dateSrv

client

getTime

getDate
Motivation

Edsko de Vries, Adrian Francalanza and Matthew Hennessy

Uniqueness Typing for MPC
Motivation
Explicit allocation and deallocation

$$\text{CLIENT}_2 \triangleq \begin{array}{l}
\text{alloc}(x).
\text{getTime}!\langle x \rangle.
x?y.
\text{getDate}!\langle x \rangle.
x?z.
\text{free } x.
\end{array}$$

$P$
Faulty server

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

Client

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

Faulty server

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

Reusing \( ret_1 \)

\[ \text{CLIENT}_1 \triangleq (\nu ret_1) \text{ getTime!}\langle ret_1 \rangle. \]
\[ ret_1?y. \]
\[ \text{getDate!}\langle ret_1 \rangle. \]
\[ ret_1?z. \]
\[ P \]
Explicit allocation and deallocation

\[ \text{CLIENT}_2 \triangleq \begin{align*}
&\text{alloc}(x). \\
&\text{getTime!}\langle x \rangle. \\
&x?y. \\
&\text{getDate!}\langle x \rangle. \\
&x?z. \\
&\text{free } x. \\
&P
\end{align*} \]
Explicit allocation and deallocation

\[ \text{CLIENT}_2 \triangleq \text{alloc}(x). \]
\[ \text{getTime}!\langle x \rangle. \]
\[ x?y. \]
\[ \text{getDate}!\langle x \rangle. \]
\[ x?z. \]
\[ \text{free } x. \]
\[ P \]
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

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

\[ a \mid \omega \text{ (unrestricted)} \]
\[ \mid 1 \text{ (affine)} \]
\[ \mid (\bullet, i) \text{ (unique after } i \text{ steps, } i \in \mathbb{N}) \]
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

\[ b_2!c \]
Initial situation

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

Diagram:

- Channel $c$ is connected to processes A, B, and C.
- Process C is connected to many other processes.

Symbols:
- $b_2$ from B to C
- $b_1$ from A to C
- $a$ from C to A

Notation:
- $c: \omega$
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 $B$

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

C sends c to both A and B

$$a!c \parallel b_1!c$$
Uniqueness

**C communicates with B on c**

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

c?x ∥ c!x

Uniqueness Typing for MPC
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

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

\text{TCon}
Type splitting

Splitting unrestricted channels

\[ [T]^{\omega} = [T]^{\omega} \circ [T]^{\omega} \]

\[ \text{PUNR} \]
Type splitting

Splitting unique channels

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

\[ \text{PUUNQ} \]

Edsko de Vries, Adrian Francalanza and Matthew Hennessy
Uniqueness Typing for MPC
35
Subtyping rule

$$\frac{\Gamma, u : T_2 \vdash P \quad T_1 \preceq_s T_2}{\Gamma, u : T_1 \vdash P} \text{TSUB}$$
Subtyping

From unrestricted to linear

\[ \omega \prec_s 1 \]

SAFF
Subtyping

From unique to unrestricted

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

\(s\)UnQ

Edsko de Vries, Adrian Francalanza and Matthew Hennessy

Uniqueness Typing for MPC
Subtyping lattice

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

Strong update

\[ \Gamma, u : [T_2]^\bullet \vdash P \]
\[ \Gamma, u : [T_1]^\bullet \vdash P \]

Type of \textit{getTime}

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

**Deallocation**

\[
\frac{\Gamma \vdash P}{\Gamma, u : [\mathcal{T}] \cdot \vdash \text{free } u. P}
\]

**Type of getDate**

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

Theorem (Type safety)

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

Theorem (Subject reduction)

If $\Gamma \models \sigma \triangleright P$ and $\sigma \triangleright P \rightarrow \sigma' \triangleright P'$ then there exists a environment $\Gamma'$ such that $\Gamma' \models \sigma' \triangleright P'$. 
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?