Uniqueness Typing for Resource Management in Message Passing Concurrency

Adrian Francalanza, Edsko de Vries and Matthew Hennessy

Synthesis, Verification, and Analysis of Rich Models
April 1-3, 2011

1The financial support of SFI is gratefully acknowledged.
Motivation

Servers

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

Client

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

timeSrv

dateSrv

client

getTime

getDateTime
Motivation

timeSrv

dateSrv

client

getTime

ret_1

getDateString
Motivation

timeSrv

dateSrv

client

getTime

getDate

ret_1

ret_2
Motivation

Recall Servers

\[
\begin{align*}
\text{TIMESRV} & \triangleq \text{rec } X. \text{getTime} \langle x \rangle. X \\
\text{DATESRV} & \triangleq \text{rec } X. \text{getDate} \langle x \rangle. X
\end{align*}
\]

Reusing \( ret_1 \)

\[
\begin{align*}
\text{CLIENT}_1 & \triangleq (\nu ret_1) \text{ getTime} \langle ret_1 \rangle. \\
& \hspace{1cm} \text{ret}_1?y. \\
& \hspace{1cm} \text{getDate} \langle ret_1 \rangle. \\
& \hspace{1cm} \text{ret}_1?z. \\
& \hspace{1cm} P
\end{align*}
\]
Motivation
Motivation
Motivation

timeSrv

dateSrv

client

getTime

getDateTime
Motivation

timeSrv

client

getDate

ret\_1

dateSrv

getTime

getDate
Motivation

Explicit deallocation

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

Explicit deallocation

\[
\text{CLIENT}_2 \triangleq (\nu \text{ret}_1) \quad \text{get\textit{Time}}!\langle \text{ret}_1 \rangle .
\]
\[
\text{ret}_1?y .
\]
\[
\text{get\textit{Date}}!\langle \text{ret}_1 \rangle .
\]
\[
\text{ret}_1?z .
\]
\[
\text{free} \ ret_1 .
\]
\[
P
\]
Runtime errors

(Faulty) timeServer

\[ \textsc{timeSrv} \triangleq \text{rec } X.\text{getTime?x.x!⟨time⟩.x!⟨time⟩.X} \]

Reusing \( ret_1 \)

\[ \textsc{client}_1 \triangleq (\nu ret_1) \text{ getTime!⟨ret_1⟩. ret}_1?y. \text{ getDate!⟨ret_1⟩. ret}_1?z. P \]
Runtime errors

(Faulty) timeServer

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

Reusing \( \text{ret}_1 \)

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

Premature deallocation

\[
\text{CLIENT}_2 \triangleq \begin{array}{l}
\text{alloc}(x).
get\ Text!\langle x\rangle.
\text{free}\ x.
x?y.
\text{getDate}!\langle x\rangle.
x?z.
P
\end{array}
\]
Purpose of this Work

- Develop a semantics for the π-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

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

\[ a ::= \omega \quad \text{(unrestricted)} \]

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

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

![Diagram showing channels and processes]

- $P$
- $Q$
- $R$
- $\cdots$

Channels $a$, $b$, $c$, and $d$ are connected as shown in the diagram.
Unrestricted channels

$R$ sends $c$ to $P$

$a!c.R'$
Unrestricted channels

\( P \) sends \( c \) to \( Q \)

\( d!c.P' \)
Linearity

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

\[ R \text{ sends } c \text{ to } P \]
\[ a!c.R' \]
Linearity

$P$ sends $c$ to $Q$

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

$Q$ communicates on $c$ with $R$

$c!v.Q'$
Linearity

\[ Q \text{ communicates on } c \text{ with } R \]
\[ c!v.Q' \]
Linearity

*R sends c to both P and Q*

\[ a!c \parallel b!c \]
Uniqueness

Initial situation

$R$ has *unique*, $(\bullet, 0)$, access to channel $c$.
Uniqueness

Initial situation

$R$ has *unique* $(\bullet, 0)$, access to channel $c$. (Note: $i = 0$!!!)
Uniqueness

*R* sends *c* to *P*

\[a!c\]
Uniqueness

$P$ sends $c$ to $Q$

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

\( Q \) communicates on \( c \) to \( R \)
\( c!v.Q \ (c \notin \text{fv}(Q)) \)
Uniqueness

$Q$ communicates on $c$ to $R$

$c!v.Q \ (c \notin \text{fv}(Q))$
Uniqueness

\( R \) sends \( c \) to both \( P \) and \( Q \)

\( a!cP_1 \parallel b!cP_2 \)
Uniqueness

\( R \) communicates with \( Q \) on \( c \)

\( c?x.P_3 \parallel c!v.R \)
Uniqueness

$R$ communicates with $P$ on $c$
$c?x.P_4 \parallel c!v.R'$
Uniqueness

d
P

Q

R
c:(\bullet, 0)

a

b

\ldots
Uniqueness

*R may transfer the uniqueness of c to P*

$a!c.P'$
Uniqueness

$R$ may dispose of $c$
free $c.P_5$
Uniqueness

*R* may ignores uniqueness and sends *c* to lots of processes
\[d!c \parallel e!c\ldots\]
Type splitting

Contraction rule

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

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

\[
\text{TCON}
\]
Type splitting

Splitting unrestricted channels

\[ [\vec{T}]_\omega = [\vec{T}]_\omega \circ [\vec{T}]_\omega \]

PU NR
Type splitting

Splitting unique channels

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

\[
\begin{array}{c}
\text{c:}(\bullet, 1) \\
\text{c:1}
\end{array}
\Rightarrow
\begin{array}{c}
\text{c:}(\bullet, 2) \\
\text{c:1 c:1}
\end{array}
\]

\[
\cdots
\]

\[
\cdots
\]
Subtyping

Subtyping rule

\[
\begin{array}{c}
\Gamma, u: T_2 \vdash P \\
T_1 \preceq_s T_2
\end{array}
\quad \frac{}{\Gamma, u: T_1 \vdash P} \quad \text{T}_\text{SUB}
\]
Subtyping

From unrestricted to linear

\[
\omega \prec_s 1 \quad \text{SAFF}
\]

\[
\omega \prec_s 1
\]

\[
\begin{array}{c}
\text{c:}\omega \\
\ldots
\end{array}
\] \Rightarrow

\[
\begin{array}{c}
\text{c:}\omega \\
\text{c:}1 \\
\ldots
\end{array}
\]
Subtyping

From unique to unrestricted

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

\[c: (\bullet, 0) \Rightarrow c: \omega\]
Subtyping lattice

\[(\bullet, 0) \rightarrow (\bullet, 1) \rightarrow (\bullet, 2) \rightarrow \ldots \rightarrow \omega \rightarrow 1\]
Usefulness of uniqueness

Strong update

\[
\frac{\Gamma, u : [T_2]^{(\bullet, 0)} \vdash P}{\Gamma \vdash T_{REV} \quad \frac{\Gamma, u : [T_1]^{(\bullet, 0)} \vdash P}{\Gamma}}
\]

Type of \textit{getTime}

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

Deallocation

\[ \begin{array}{c}
\Gamma \vdash P \\
\Gamma, u : [T]^{(\bullet,0)} \vdash \text{free } u \cdot P \\
\end{array} \]

Type of \textit{getDate}

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

▶ Equational Reasoning about well-typed processes.
▶ Efficiency Reasoning about well-typed processes.
▶ Extend static analysis to the Higher-Order $\pi$-calculus.
References

Uniqueness Typing for Resource Management in Message Passing Concurrency,
*Linerity 2009*. (Journal version submitted for publication)

Reasoning about Resource Management for Message Passing Concurrency,
*Places 2011*. 