# Uniqueness Typing for Resource Management in Message Passing Concurrency

Adrian Francalanza, Edsko de Vries[1] and Matthew Hennessy[1]

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

[1]The financial support of SFI is gratefully acknowledged.

# Motivation

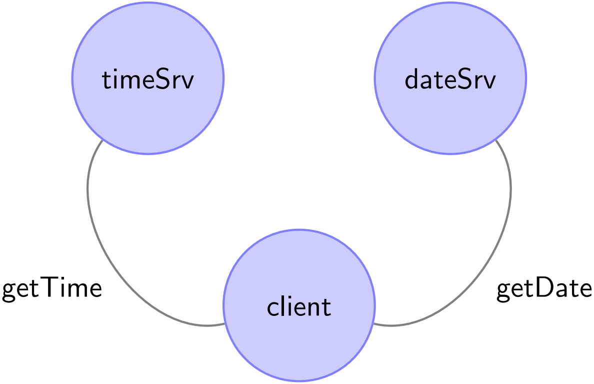
## Servers

$$\begin{array}{lcl} \textsc{timeSrv} & \triangleq & \mathsf{rec}\, X.getTime?x.x!\langle time\rangle.X \\ \textsc{dateSrv} & \triangleq & \mathsf{rec}\, X.getDate?x.x!\langle date\rangle.X \end{array}$$

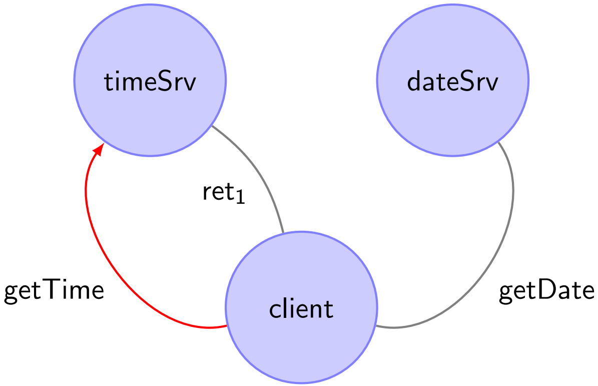
## Client

$$\begin{array}{lcll} \textsc{client}_0 & \triangleq & (\nu\, ret_1) & getTime!\langle ret_1\rangle. \\ & & & ret_1?y. \\ & & (\nu\, ret_2) & getDate!\langle ret_2\rangle. \\ & & & ret_2?z. \\ & & & P \end{array}$$

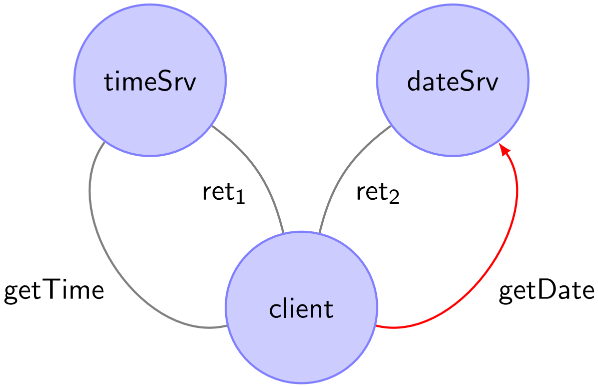
# Motivation



# Motivation



# Motivation



# Motivation

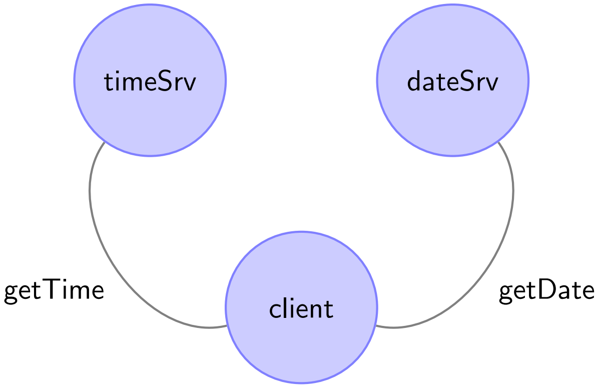
## Recall Servers

$$\begin{aligned}\textsc{timeSrv} &\triangleq \mathsf{rec}\, X.getTime?x.\ x!\langle time\rangle.X \\ \textsc{dateSrv} &\triangleq \mathsf{rec}\, X.getDate?x.\ x!\langle date\rangle.X\end{aligned}$$

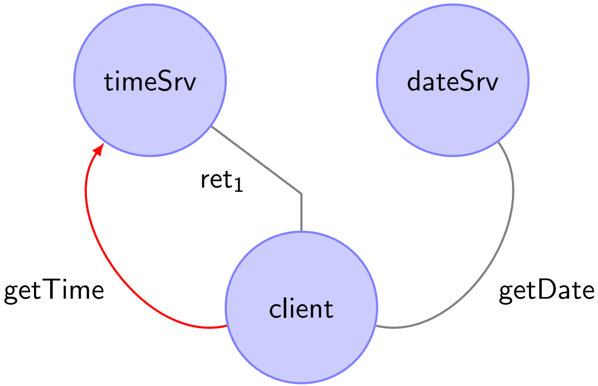
## Reusing $ret_1$

$$\begin{aligned}\textsc{client}_1 \triangleq (\nu ret_1)\ & getTime!\langle ret_1\rangle. \\ & ret_1?y. \\ & getDate!\langle ret_1\rangle. \\ & ret_1?z. \\ & P\end{aligned}$$

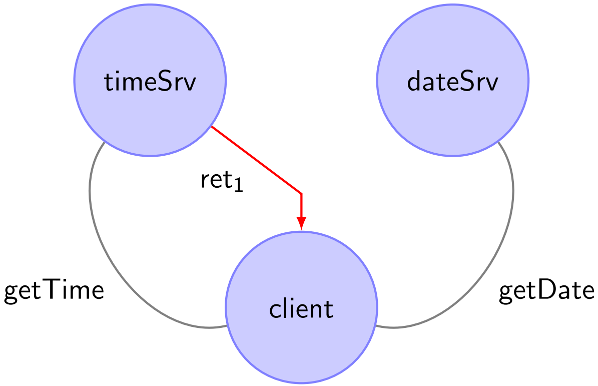
# Motivation



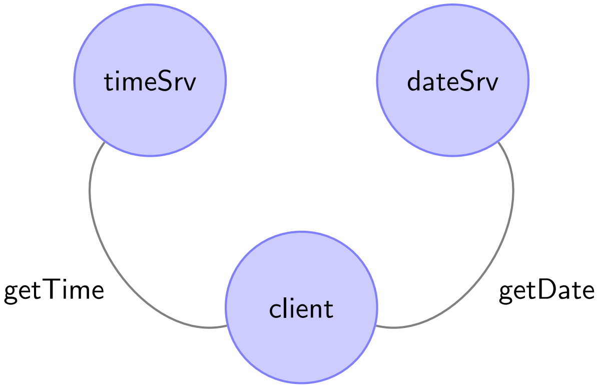
# Motivation



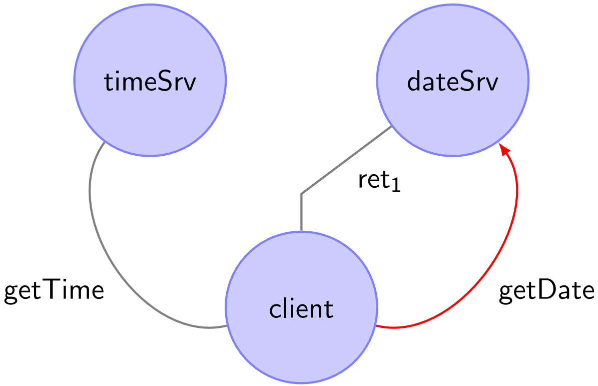
# Motivation



# Motivation



# Motivation



# Motivation

## Explicit deallocation

$$
\begin{aligned}
\textsc{Client}_2 \quad \triangleq \quad (\nu\, ret_1) \quad & getTime!\langle ret_1 \rangle. \\
& ret_1?y. \\
& getDate!\langle ret_1 \rangle. \\
& ret_1?z. \\
& \mathsf{free}\ ret_1. \\
& P
\end{aligned}
$$

# Motivation

## Explicit deallocation

$$\begin{array}{llll}\textsc{Client}_2 & \triangleq & (\nu\, ret_1) & getTime!\langle ret_1\rangle. \\ & & & ret_1?y. \\ & & & getDate!\langle ret_1\rangle. \\ & & & ret_1?z. \\ & & & \mathsf{free}\ ret_1. \\ & & & P\end{array}$$

# Runtime errors

## (Faulty) timeServer

$$\textsc{timeSrv} \triangleq \mathsf{rec}\, X.getTime?x.x!\langle time\rangle.x!\langle time\rangle.X$$

## Reusing $ret_1$

$$\begin{array}{lll} \textsc{client}_1 & \triangleq & (\nu ret_1) \quad getTime!\langle ret_1\rangle. \\ & & \qquad\quad ret_1?y. \\ & & \qquad\quad getDate!\langle ret_1\rangle. \\ & & \qquad\quad ret_1?z. \\ & & \qquad\quad P \end{array}$$

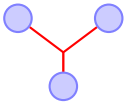
# Runtime errors

## (Faulty) timeServer

$$\textsc{timeSrv} \triangleq \mathsf{rec}\, X.\mathit{getTime}?x.\color{red}{x!\langle \mathit{time}\rangle.x!\langle \mathit{time}\rangle.}\color{black}X$$

## Reusing $\mathit{ret}_1$

$$\begin{aligned}\textsc{client}_1 \triangleq (\nu\, \mathit{ret}_1) \quad & \mathit{getTime}!\langle \mathit{ret}_1\rangle.\\ & \mathit{ret}_1?y.\\ & \mathit{getDate}!\langle \mathit{ret}_1\rangle.\\ & \mathit{ret}_1?z.\\ & P\end{aligned}$$



# Runtime errors

## Premature deallocation

$$\textsc{Client}_2 \triangleq \begin{array}{l} \mathsf{alloc}(x). \\ getTime!\langle x\rangle. \\ \mathsf{free}\, x. \\ x?y. \\ getDate!\langle x\rangle. \\ x?z. \\ P \end{array}$$

# Purpose of this Work

- 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.

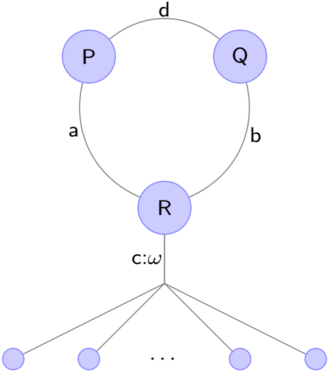
# Type language

$$
\begin{array}{llll}
\mathbf{T} & ::= & [\mathbf{T}]^{a} & \text{(channel type)} \\
 & & & \\
a & ::= & \omega & \text{(unrestricted)} \\
 & \mid & 1 & \text{(affine)} \\
 & \mid & (\bullet, i) & \text{(unique after } i \text{ steps, } i \in \mathbb{N}\text{)}
\end{array}
$$

# Unrestricted channels

## Initial situation

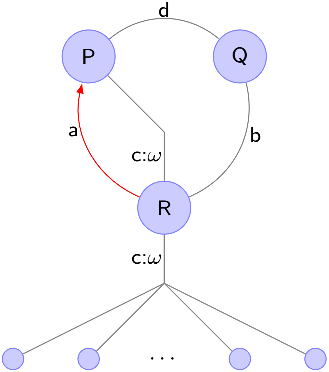
Channel $c$ is unrestricted ($\omega$) and shared by many processes



# Unrestricted channels

*R* sends *c* to *P*

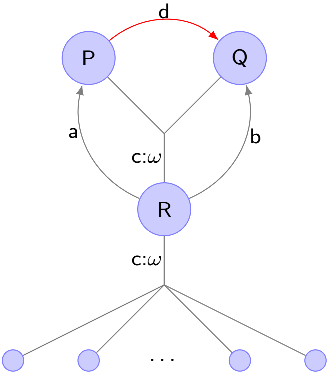
$a!c.R'$



# Unrestricted channels

$P$ sends $c$ to $Q$

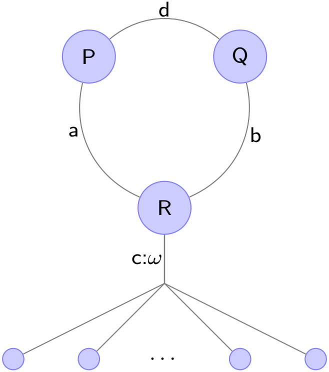
$d!c.P'$



# Linearity

## Initial situation

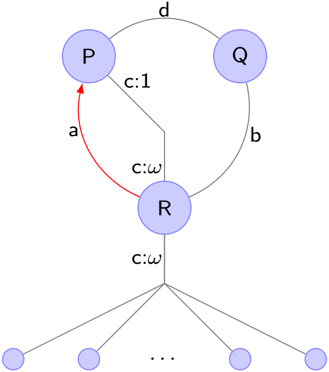
Channel $c$ is unrestricted ($\omega$) and shared by many processes



# Linearity

*R* sends *c* to *P*

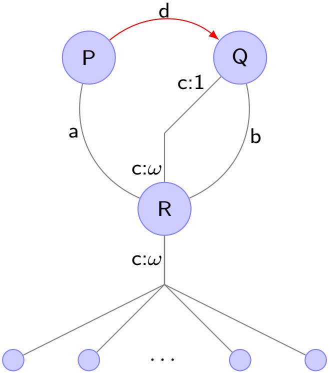
$a!c.R'$



# Linearity

*P* sends *c* to *Q*

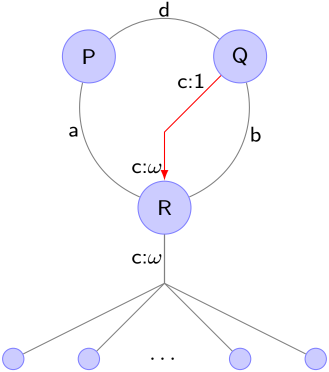
$d!c.P' \; (c \notin \mathsf{fv}\,(P'))$



# Linearity

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

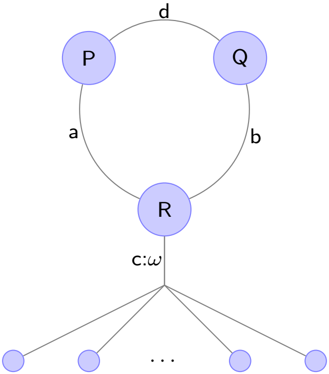
$c!v.Q'$



# Linearity

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

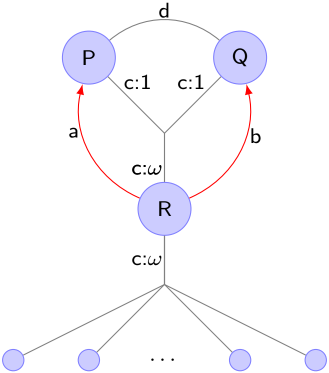
$c!v.Q'$



# Linearity

*R* sends *c* to both *P* and *Q*

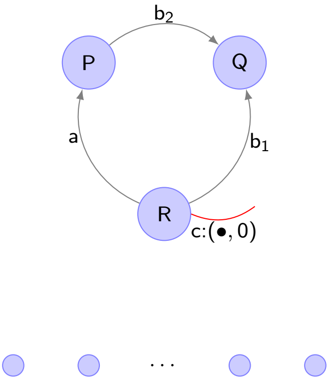
$a!c \parallel b!c$



# Uniqueness

## Initial situation

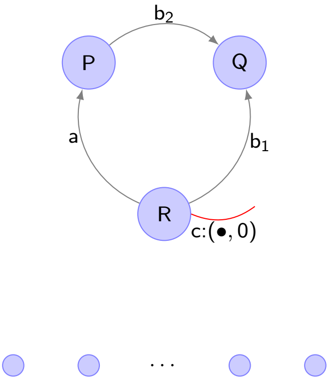
$R$ has *unique*, $(\bullet, 0)$, access to channel $c$.



# Uniqueness

## Initial situation

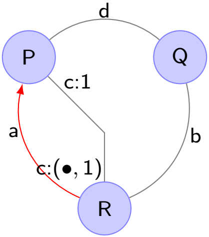
$R$ has *unique*, $(\bullet, 0)$, access to channel $c$. (Note: $i = 0$!!!)

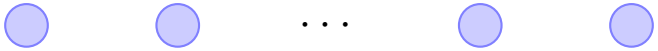


# Uniqueness

*R* sends *c* to *P*

$a!c$

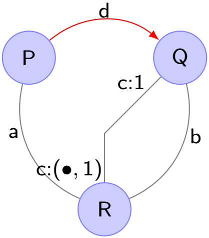


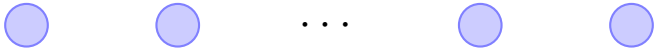



# Uniqueness

*P* sends *c* to *Q*

$d!c.P' \ (c \notin \mathrm{fv}\,(P'))$

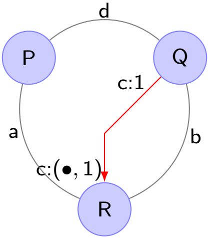


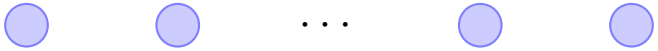



# Uniqueness

*Q* communicates on *c* to *R*

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

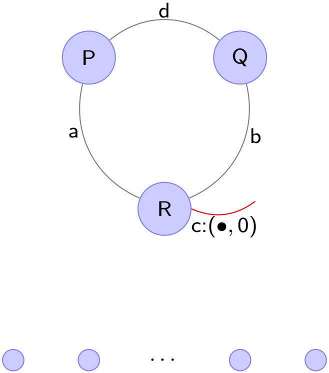


# Uniqueness

*Q* communicates on *c* to *R*

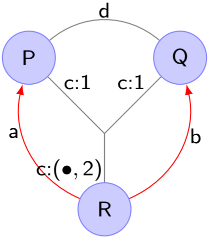
$c!v.Q\ (c \notin \mathrm{fv}(Q))$

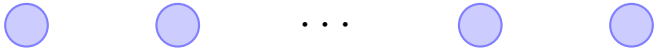


# Uniqueness

*R* sends *c* to both *P* and *Q*

$a!cP_1 \parallel b!cP_2$

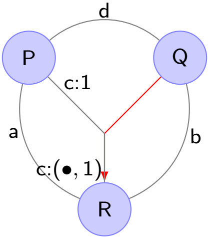


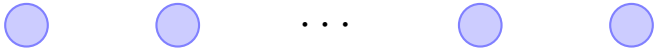



# Uniqueness

*R* communicates with *Q* on *c*

$c?x.P_3 \parallel c!v.R$

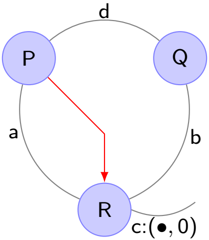


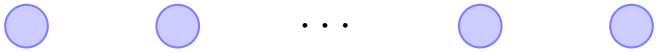



# Uniqueness

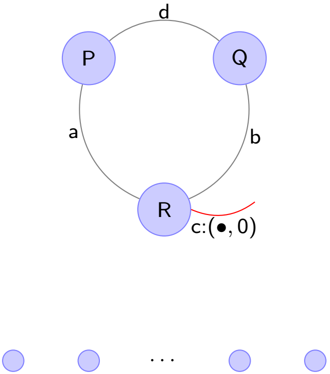
$R$ communicates with $P$ on $c$

$c?x.P_4 \parallel c!v.R'$



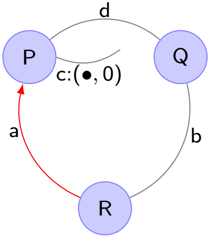
# Uniqueness

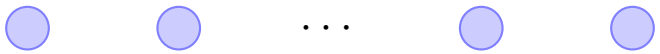


# Uniqueness

$R$ may transfer the uniqueness of $c$ to $P$

$a!c.P'$

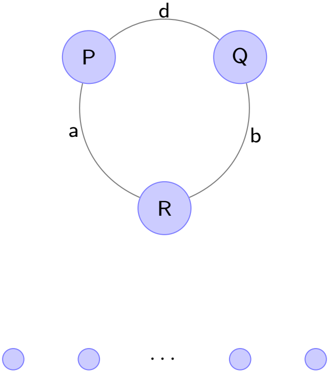


# Uniqueness

## $R$ may dispose of $c$

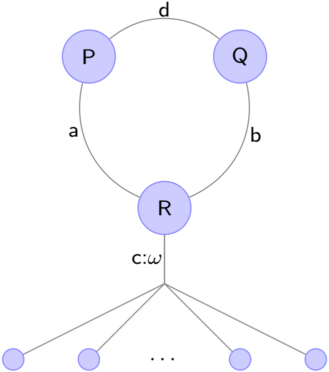
free $c.P_5$



# Uniqueness

*R* may ignores uniqueness and sends *c* to lots of processes

$d!c \parallel e!c \ldots$



# Type splitting

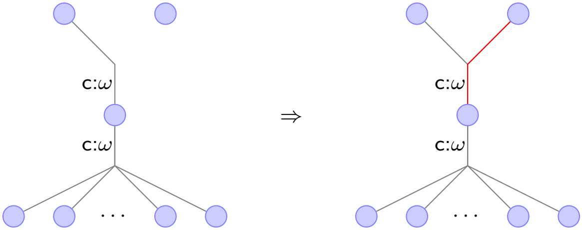
## Contraction rule

$$\frac{\Gamma, u\!:\!\mathbf{T}_1, u\!:\!\mathbf{T}_2 \vdash P \quad \mathbf{T} = \mathbf{T}_1 \circ \mathbf{T}_2}{\Gamma, u\!:\!\mathbf{T} \vdash P}\ \textsc{tCon}$$

# Type splitting

## Splitting unrestricted channels

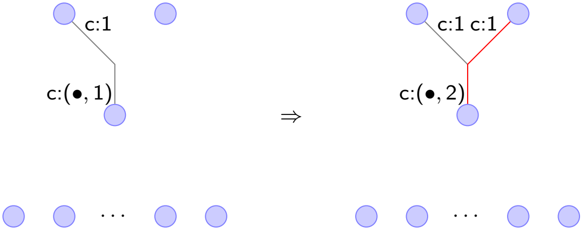
$$\frac{}{[\vec{\mathbf{T}}]^\omega = [\vec{\mathbf{T}}]^\omega \circ [\vec{\mathbf{T}}]^\omega}\ \textsc{pUnr}$$



# Type splitting

## Splitting unique channels

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



# Subtyping

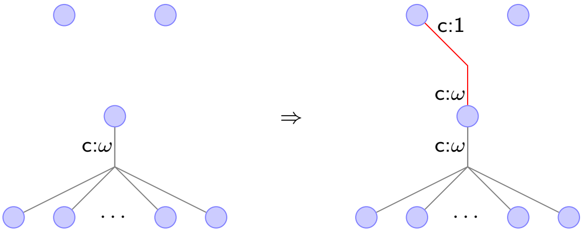
## Subtyping rule

$$\frac{\Gamma, u : \mathbf{T}_2 \vdash P \quad \mathbf{T}_1 \prec_s \mathbf{T}_2}{\Gamma, u : \mathbf{T}_1 \vdash P}\ \textsc{tSub}$$

# Subtyping

## From unrestricted to linear

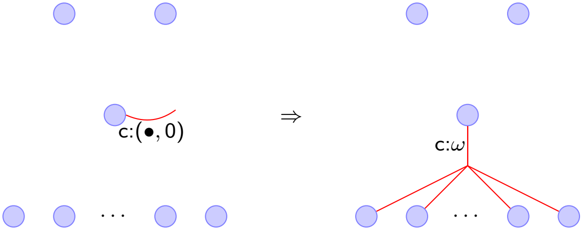
$$\frac{}{\omega \prec_s 1}\ \textsc{sAff}$$



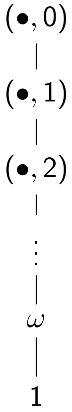
# Subtyping

## From unique to unrestricted

$$\frac{}{(\bullet, i) \prec_s \omega}\ \textsc{sUnq}$$



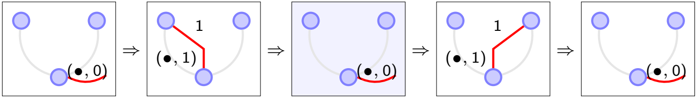
# Subtyping lattice



# Usefulness of uniqueness

## Strong update

$$\frac{\Gamma, u : [\mathbf{T}_2]^{(\bullet,0)} \vdash P}{\Gamma, u : [\mathbf{T}_1]^{(\bullet,0)} \vdash P} \textsc{tRev}$$



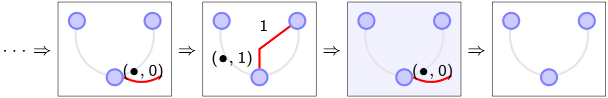

## Type of *getTime*

$getTime : [[\mathbf{Time}]^1]^\omega$

# Usefulness of uniqueness

## Deallocation

$$\frac{\Gamma \vdash P}{\Gamma, u : [\mathbf{T}]^{(\bullet,0)} \vdash \text{free } u.P} \textsc{tFree}$$




## Type of *getDate*

$getDate : [[\mathbf{Date}]^1]^\omega$

# 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* $\Sigma \triangleright P \longrightarrow \Sigma' \triangleright P'$ *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

[1] E. DeVries, A. Francalanza, M. Hennessy
Uniqueness Typing for Resource Management in Message Passing Concurrency,
*Linerity 2009.* (Journal version submitted for publication)

[2] E. DeVries, A. Francalanza, M. Hennessy
Reasoning about Resource Management for Message Passing Concurrency,
*Places 2011.*