2019 Brazilian Symposium on Formal Methods (SBMF)

A Polymorphic RPC Calculus

Kwanghoon Choi

Chonnam National University, Republic of Korea

November 28, 2019

(Joint work with James Cheney, Simon Fowler, and Sam Lindley,
University of Edinburgh, UK)

Introduction

For a web system, a server program and a client program are developed separately but needed to be tested/maintained together.

Tierless programming languages for client-server model such as Web

- Programmers can write client and server expressions in a single program without worrying about complex communications
- Compilers will automatically slice the program into two parts which run on the server and on the client, respectively.

Introduction (Cont.)

1. The RPC calculus λ_{rpc} : the simple semantics foundation for the tierless programming languages (Cooper&Wadler, 2009)

2. The typed RPC calculus λ_{rpc}^{typed}: location information such as `client` and `server` in types (Choi&Chang, 2019)

3. The polymorphic RPC calculus $\lambda_{rpc}^{\forall}$: polymorphic locations (This paper and its extension)

	(*) Dynamic check	Location type	Slicing
RPC	always	(untyped)	yes
Typed RPC	never	monomorphic loc.	yes
Poly. RPC	never/only for poly loc.	polymorphic loc.	yes

(*): Dynamic location checking on local/remote procedure calls

cf. The other tierless PLs: Eliom, Hop, ML5, Ur/Web, and so on.

The RPC calculus λ_{rpc} (Cooper&Wadler, 2009)

The simple semantics foundation for the tierless programming languages using λ-applications for remote/local procedure calls

```
fun main() client { authenticate () }

fun authenticate() server {
 var creds = getCredentials( "Enter name:passwd > " )
 if ( creds=="ezra:opensesame" ) { "The secret document" }
 else { "Access denied" }
}

fun getCredentials(prompt) client {
 (print(prompt);  read)
}
```

The RPC calculus λ_{rpc} (Cont.)

A call-by-value λ-calculus with location annotated λ-abstractions

- $\lambda^a x.N$: λ-abstractions that must run at location a
- $M \Downarrow_a V$: A term M at location a evaluates to V.

$$\begin{array}{llcll} \text{Location} & a, b & ::= & \mathbf{c} & \mid \quad \mathbf{s} \\ \\ \text{Term} & L, M, N & ::= & V & \mid \quad L\ M \\ \text{Value} & V, W & ::= & x & \mid \quad \lambda^a x.N \end{array}$$

Evaluation

$$\frac{}{\lambda^b x.M \Downarrow_a \lambda^b x.M}\ (\text{Abs})$$

$$\frac{L \Downarrow_a \lambda^b x.N \qquad M \Downarrow_a W \qquad N\{W/x\} \Downarrow_b V}{L\ M \Downarrow_a V}\ (\text{App})$$

Note that every procedure call should do a dynamic location check if it is a remote procedure call or not.

The typed RPC calculus λ_{rpc}^{typed} (Choi&Chang 2019)

Location-annotated function types $A \xrightarrow{a} B$

(1) Every λ-abstraction of type $A \xrightarrow{a} B$ runs at location a.

Type $A, B ::= base \mid A \xrightarrow{a} B$

- $(\lambda^{\mathbf{s}} f.\ (\lambda^{\mathbf{s}} x.x)\ (f\ M))\ \ (\lambda^{\mathbf{c}} y.\ (\lambda^{\mathbf{s}} z.z)\ \ y)$

  Well-typed where $f : A \xrightarrow{\mathbf{c}} B$

- $(\ \lambda^{\mathbf{c}} f.\ f\ M\ )\ \ (\text{if } \cdots \text{ then } \lambda^{\mathbf{c}}\text{x.M}_1 \text{ else } \lambda^{\mathbf{s}}\text{y.M}_2)$

  Ill-typed because neither $f : A \xrightarrow{\mathbf{c}} B$ nor $f : A \xrightarrow{\mathbf{s}} B$

(2) A typing judgment, $\Gamma \vdash_a M : A$, says:

- A term M at location a has type A under a type environment Γ

The typed RPC calculus λ_{rpc}^{typed} (Cont.)

The key idea: A refinement of the lambda application typing w.r.t. the two locations a and b

$$\frac{\Gamma \vdash_a L : A \xrightarrow{b} B \qquad \Gamma \vdash_a M : A}{\Gamma \vdash_a L\ M : B}$$

Our analysis using caller location(a) and callee location(b):

- $a = b : L\ M$ is a local procedure call
- $a = \mathbf{c}$ and $b = \mathbf{s} : L\ M$ is an RPC from the client to the server
- $a = \mathbf{s}$ and $b = \mathbf{c} : L\ M$ is an RPC from the server to the client

By the type soundness, it is guaranteed that every remote procedure call thus analyzed statically will not be changed into any local procedure calls in runtime.

The typed RPC calculus λ_{rpc}^{typed} (Cont.)

A slicing compilation of λ_{rpc}^{typed} into λ_{cs}.

$$\frac{\Gamma \vdash_a L : A \xrightarrow{b} B \qquad \Gamma \vdash_a M : A}{\Gamma \vdash_a L\ M : B}$$

Caller(a), Callee(b)	λ_{rpc}^{typed}	λ_{cs}	Procedure call
$a = b$	$L\ M$	$L(M)$	Local ($\mathbf{c}\rightarrow\mathbf{c}$ or $\mathbf{s}\rightarrow\mathbf{s}$)
$a = \mathbf{c}$ and $b = \mathbf{s}$	$L\ M$	$req(L, M)$	Remote ($\mathbf{c}\rightarrow\mathbf{s}$)
$a = \mathbf{s}$ and $b = \mathbf{c}$	$L\ M$	$call(L, M)$	Remote ($\mathbf{s}\rightarrow\mathbf{c}$)

For example, req(V,W), can be implemented by HTTP request while call(V,W) can be implemented by HTTP response.

In λ_{cs}, no dynamic checking on locations to make a decision about local/remote procedure calls is required in run-time.

The typed RPC calculus λ_{rpc}^{typed} (Cont.)

A slicing compilation example of λ_{rpc}^{typed} into λ_{cs}.

$$(\lambda^{\mathbf{s}} f.\ (\lambda^{\mathbf{s}} x.x)\ {}^{\mathbf{s}_3}(f\ {}^{\mathbf{c}_1} c))\ {}^{\mathbf{s}_1}(\lambda^{\mathbf{c}} y.\ (\lambda^{\mathbf{s}} z.z)\ {}^{\mathbf{s}_2} y)$$

$$\Rightarrow$$

$$\begin{array}{lll}
\phi_{\mathbf{c}}: & \mathit{main} & = \ \text{let } r_3 = \text{req}_{s1}(\mathit{clo}(g_3, \{\}),\ \mathit{clo}(g_5, \{\})) \text{ in } r_3 \\
& g_2 & = \{f_7\}\ \lambda^{\mathbf{c}} z_{10}.\ \text{let } y_9 = f_7\ z_{10} \text{ in } \text{ret}(y_9) \\
& g_5 & = \{\}\ \lambda^{\mathbf{c}} y.\ \text{let } r_{14} = \text{req}_{s2}(\mathit{clo}(g_4, \{\}),\ y) \text{ in } r_{14} \\
\\
\phi_{\mathbf{s}}: & g_1 & = \{\}\ \lambda^{\mathbf{s}} x.\ x \\
& g_3 & = \{\}\ \lambda^{\mathbf{s}} f.\ \text{let } x_5 = (\text{let } r_{11} = \text{call}_{c1}(\mathit{clo}(g_2, \{f\}),\ c) \text{ in } r_{11}) \text{ in} \\
& & \qquad \text{let } r_6 = \mathit{clo}(g_1, \{\})\ {}_{s3}\ x_5 \text{ in } r_6 \\
& g_4 & = \{\}\ \lambda^{\mathbf{s}} z.\ z
\end{array}$$

Motivation

The typed RPC calculus only uses monomorphic locations $\mathbf{c}$ and $\mathbf{s}$, and so it does not support polymorphic location functions.

- For example, map_{poly} cannot be written in λ_{rpc}^{typed}.
- Instead, we should write map_{client} and map_{server} separately.

Some limited forms of polymorphic locations have already been used in Links and Eliom. But there is no semantics foundation yet.

Even after polymorphic locations are introduced, we still want to identify local/remote procedure calls statically.

Motivation (Cont.)

Our approach: *Parametric location polymorphism*

- *Location abstraction* '$\Lambda l.M$' where l is a location variable.

$$poly \;=\; (\Lambda l.\lambda^{l} f.\lambda^{l} xs.\; \cdots) \;:\; \forall l.(A \xrightarrow{l} B) \xrightarrow{l} ([A] \xrightarrow{l} [B])$$

- *Location application* '$M[Loc]$' where Loc is a location constant, a, or a variable, l.

$$poly[\mathbf{c}] \;=\; (\lambda^{\mathbf{c}} f.\lambda^{\mathbf{c}} xs.\; \cdots) \;:\; (A \xrightarrow{\mathbf{c}} B) \xrightarrow{\mathbf{c}} ([A] \xrightarrow{\mathbf{c}} [B])$$

A technical problem

In the presence of location variables, the previous key idea in λ_{rpc}^{typed} might not work so well as:

$$\frac{\Gamma \vdash_{l_1} L : A \xrightarrow{l_2} B \qquad \Gamma \vdash_{l_1} M : A}{\Gamma \vdash_{l_1} L\ M : B}$$

$\Rightarrow$ Two location variables l_1 and l_2 could not be compared easily to determine statically if $L\ M$ is local or remote procedure calls.

Two approaches to polymorphic locations

- A static approach: A monomorphization translation into λ_{rpc}^{typed} [This paper, SBMF 2019]
- A dynamic approach: An extended client-server calculus with location polymorphic applications, $gen(Loc, f, arg)$ [Extension]

A solution for $\lambda^{\forall}_{rpc}$ and its slicing compilation

(1) Polymorphic locations are supported by a translation into monomorphic ones

How? by interpreting

- location abstraction as a pair of client/server instances, and
- location application as a projection from the pair

For example,

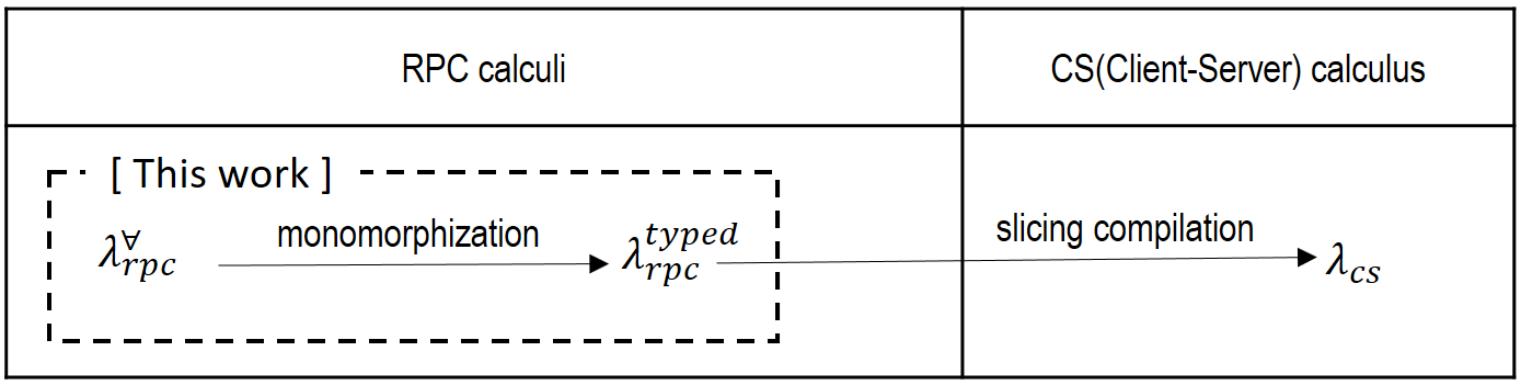
$$\begin{aligned} id = \Lambda l.\lambda^{l} x.x &\Rightarrow [\![id]\!] = (\lambda^{\mathbf{c}} x.x, \lambda^{\mathbf{s}} x.x) \\ id[\mathbf{c}] &\Rightarrow \pi_1([\![id]\!]) \\ id[\mathbf{s}] &\Rightarrow \pi_2([\![id]\!]) \end{aligned}$$

(2) After the translation, the existing λ^{typed}_{rpc} slicing compilations can be fully utilized.

Our contribution: $\lambda^{\forall}_{rpc}$

A polymorphic RPC calculus as a conservative extension of λ^{typed}_{rpc}

- A polymorphic RPC calculus that supports a parametric polymorphism over locations
- A monomorphization translation of the polymorphic RPC calculus into the typed RPC calculus
- Type soundness of the type system and the type/semantics correctness of the translation



Part I: A polymorphic RPC calculus $\lambda^{\forall}_{rpc}$

Location	a, b	$::=$	$\mathbf{c}$	$\mid$	$\mathbf{s}$			
	Loc	$::=$	a	$\mid$	l			
Term	L, M, N	$::=$	V	$\mid$	$L\ M$	$\mid$	$M[Loc]$	$\mid$ $M[A]$
Value	V, W	$::=$	x	$\mid$	$\lambda^{Loc} x.N$	$\mid$	$\Lambda l.V$	$\mid$ $\Lambda\alpha.V$

Evaluation

$$\frac{}{\lambda^b x.M \Downarrow_a \lambda^b x.M}\ (\text{Abs})$$

$$\frac{L \Downarrow_a \lambda^b x.N \qquad M \Downarrow_a W \qquad N\{W/x\} \Downarrow_b V}{L\ M \Downarrow_a V}\ (\text{App})$$

$$\frac{}{\Lambda l.V \Downarrow_a \Lambda l.V}\ (\text{Labs}) \qquad \frac{M \Downarrow_a \Lambda l.V}{M[b] \Downarrow_a V\{b/l\}}\ (\text{Lapp})$$

cf. (Tabs) and (Tapp)

A type system for the polymorphic RPC calculus

Location	a, b	$::=$	$\mathbf{c}$	$\mid$	$\mathbf{s}$				
	Loc	$::=$	a	$\mid$	l				
Type	A, B	$::=$	base	$\mid$	$A \xrightarrow{\mathit{Loc}} A$	$\mid$	$\forall l.A$	$\mid$ α	$\mid$ $\forall\alpha.A$

Typing rules

$$\text{(T-Var)}\ \frac{\Gamma(x) = A}{\Gamma \vdash_{\mathit{Loc}} x : A} \qquad \text{(T-Abs)}\ \frac{\Gamma\{x : A\} \vdash_{\mathit{Loc}} M : B}{\Gamma \vdash_{\mathit{Loc}'} \lambda^{\mathit{Loc}} x.M : A \xrightarrow{\mathit{Loc}} B}$$

$$\text{(T-App)}\ \frac{\Gamma \vdash_{\mathit{Loc}} L : A \xrightarrow{\mathit{Loc}'} B \qquad \Gamma \vdash_{\mathit{Loc}} M : A}{\Gamma \vdash_{\mathit{Loc}} L\ M : B}$$

$$\text{(T-Labs)}\ \frac{\Gamma, l \vdash_{\mathit{Loc}} V : A}{\Gamma \vdash_{\mathit{Loc}} \Lambda l.V : \forall l.A} \qquad \text{(T-Lapp)}\ \frac{\Gamma \vdash_{\mathit{Loc}} M : \forall l.A}{\Gamma \vdash_{\mathit{Loc}} M[\mathit{Loc}'] : A\{\mathit{Loc}'/l\}}$$

cf. (T-Tabs) and (T-Tapp)

Properties of the polymorphic type system

Type soundness for the polymorphic RPC calculus

- For a closed term M, if $\emptyset \vdash_a M : A$ and $M \Downarrow_a V$, then $\emptyset \vdash_a V : A$.

The polymorphic RPC calculus can be viewed as a conservative extension of the typed RPC calculus.

- Regardless of the introduction of polymorphic locations, every remote procedure call identified statically will remain as RPC during evaluation.

Part II: A monomorphization translation of $\lambda^{\forall}_{rpc}$ into λ^{typed}_{rpc}

A basic idea: location abstraction as a pair of client/server instances, and location application as its projection

Translation: types

$[\![\mathit{base}]\!] = \mathit{base}$

$[\![\alpha]\!] = \alpha$ $\qquad$ $[\![\forall\alpha.A]\!] = \forall\alpha.[\![A]\!]$

$[\![A \xrightarrow{a} B]\!] = [\![A]\!] \xrightarrow{a} [\![B]\!]$ $\qquad$ $[\![\forall l.A]\!] = [\![A\{\mathbf{c}/l\}]\!] \times [\![A\{\mathbf{s}/l\}]\!]$

Translation: terms

$[\![x]\!] = x$ $\qquad$ $[\![\lambda^a x.M]\!] = \lambda^a x.[\![M]\!]$

$[\![L\ M]\!] = [\![L]\!]\ [\![M]\!]$

$[\![\Lambda\alpha.V]\!] = \Lambda\alpha.[\![V]\!]$ $\qquad$ $[\![M[B]]\!] = [\![M]\!][[\![B]\!]]$

$[\![\Lambda l.V]\!] = ([\![V\{\mathbf{c}/l\}]\!], [\![V\{\mathbf{s}/l\}]\!])$

$[\![M[\mathbf{c}]]\!] = \pi_1([\![M]\!])$ $\qquad$ $[\![M[\mathbf{s}]]\!] = \pi_2([\![M]\!])$

Example (1)

The identity function:

$$
\begin{aligned}
[\![\forall I.\forall\alpha.\alpha \xrightarrow{I} \alpha]\!] &= ([\![(\forall\alpha.\alpha \xrightarrow{I} \alpha)\{\mathbf{c}/I\}]\!],\ [\![(\forall\alpha.\alpha \xrightarrow{I} \alpha)\{\mathbf{s}/I\}]\!]) \\
&= ([\![(\forall\alpha.\alpha \xrightarrow{\mathbf{c}} \alpha)]\!],\ [\![(\forall\alpha.\alpha \xrightarrow{\mathbf{s}} \alpha)]\!]) \\
&= (\forall\alpha.[\![(\alpha \xrightarrow{\mathbf{c}} \alpha)]\!],\ \forall\alpha.[\![(\alpha \xrightarrow{\mathbf{s}} \alpha)]\!]) \\
&= (\forall\alpha.[\![\alpha]\!] \xrightarrow{\mathbf{c}} [\![\alpha]\!],\ \forall\alpha.[\![\alpha]\!] \xrightarrow{\mathbf{s}} [\![\alpha]\!]) \\
&= (\forall\alpha.\alpha \xrightarrow{\mathbf{c}} \alpha,\ \forall\alpha.\alpha \xrightarrow{\mathbf{s}} \alpha)
\end{aligned}
$$

$$
\begin{aligned}
[\![\Lambda I.\Lambda\alpha.\lambda^{I}x.x]\!] &= ([\![(\Lambda\alpha.\lambda^{I}x.x)\{\mathbf{c}/I\}]\!],\ [\![(\Lambda\alpha.\lambda^{I}x.x)\{\mathbf{s}/I\}]\!]) \\
&= ([\![\Lambda\alpha.\lambda^{\mathbf{c}}x.x]\!],\ [\![\Lambda\alpha.\lambda^{\mathbf{s}}x.x]\!]) \\
&= (\Lambda\alpha.[\![\lambda^{\mathbf{c}}x.x]\!],\ \Lambda\alpha.[\![\lambda^{\mathbf{s}}x.x]\!]) \\
&= (\Lambda\alpha.\lambda^{\mathbf{c}}x.[\![x]\!],\ \Lambda\alpha.\lambda^{\mathbf{s}}x.[\![x]\!]) \\
&= (\Lambda\alpha.\lambda^{\mathbf{c}}x.x,\ \Lambda\alpha.\lambda^{\mathbf{s}}x.x).
\end{aligned}
$$

Example (2)

A map function of type $\forall I.(A \xrightarrow{I} B) \xrightarrow{I} ([A] \xrightarrow{I} [B])$:

$$\texttt{letrec} \quad map = \Lambda I.\lambda^I f.\lambda^I xs.M_I \quad \texttt{in} \quad \cdots$$

where $M_X \stackrel{\text{def}}{=\!=} case\ xs\ of\ \{[] \Rightarrow []; \ (y : ys) \Rightarrow f\ \ y :: map[X]\ \ f\ \ ys\}$

$$\begin{aligned}
[\![map]\!] &= ([\![(\lambda^I f.\lambda^I xs.M_I)\{\mathbf{c}/I\}]\!], \ \ [\![(\lambda^I f.\lambda^I xs.M_I)\{\mathbf{s}/I\}]\!]) \\
&= ([\![\lambda^{\mathbf{c}} f.\lambda^{\mathbf{c}} xs.M_{\mathbf{c}}]\!], \ \ [\![\lambda^{\mathbf{s}} f.\lambda^{\mathbf{s}} xs.M_{\mathbf{s}}]\!]) \\
&= (\lambda^{\mathbf{c}} f.[\![\lambda^{\mathbf{c}} xs.M_{\mathbf{c}}]\!], \ \ \lambda^{\mathbf{s}} f.[\![\lambda^{\mathbf{s}} xs.M_{\mathbf{s}}]\!]) \\
&= (\lambda^{\mathbf{c}} f.\lambda^{\mathbf{c}} xs.[\![M_{\mathbf{c}}]\!], \ \ \lambda^{\mathbf{s}} f.\lambda^{\mathbf{s}} xs.[\![M_{\mathbf{s}}]\!]).
\end{aligned}$$

where

- $[\![map[\mathbf{c}]]\!]$ in $[\![M_{\mathbf{c}}]\!]$ is translated as $\pi_1([\![map]\!])$, and
- $[\![map[\mathbf{s}]]\!]$ in $[\![M_{\mathbf{s}}]\!]$ is translated as $\pi_2([\![map]\!])$.

Example (3)

A map function of type $\forall l_1.\forall l_2.\forall l_3.(A \xrightarrow{l_3} B) \xrightarrow{l_1} ([A] \xrightarrow{l_2} [B])$:

$$\texttt{letrec} \quad map = \Lambda l_1.\Lambda l_2.\Lambda l_3.map_0 \quad \texttt{in} \quad \cdots$$

where $map_0 \;=\; \lambda^{l_1} f.\lambda^{l_2} xs.\; M_{l_1 l_2 l_3}$

$[\![map]\!]$ is translated as

$$\left(\begin{array}{l} \left(\begin{array}{l} ([\![map_0\{\mathbf{c}/l_1,\mathbf{c}/l_2,\mathbf{c}/l_3\}]\!],[\![map_0\{\mathbf{c}/l_1,\mathbf{c}/l_2,\mathbf{s}/l_3\}]\!]), \\ ([\![map_0\{\mathbf{c}/l_1,\mathbf{s}/l_2,\mathbf{c}/l_3\}]\!],[\![map_0\{\mathbf{c}/l_1,\mathbf{s}/l_2,\mathbf{s}/l_3\}]\!]) \end{array}\right), \\ \left(\begin{array}{l} ([\![map_0\{\mathbf{s}/l_1,\mathbf{c}/l_2,\mathbf{c}/l_3\}]\!],[\![map_0\{\mathbf{s}/l_1,\mathbf{c}/l_2,\mathbf{s}/l_3\}]\!]), \\ ([\![map_0\{\mathbf{s}/l_1,\mathbf{s}/l_2,\mathbf{c}/l_3\}]\!],[\![map_0\{\mathbf{s}/l_1,\mathbf{s}/l_2,\mathbf{s}/l_3\}]\!]) \end{array}\right) \end{array}\right)$$

The monomorphization translation could generate many instances in theory, but the first map function would be common in practice.

$\Rightarrow$ The dynamic approach can solve the problem at the cost of passing locations and checking them in run-time.

Properties of the monomorphization translation

Type correctness of the translation

- For a closed term M, if $\emptyset \vdash_a M : A$ in $\lambda^{\forall}_{rpc}$ then $\emptyset \vdash_a [\![M]\!] : [\![A]\!]$ in λ^{typed}_{rpc}.

Semantic correctness of the translation

- For a closed term M, if $\emptyset \vdash_a M : A$ and $M \Downarrow_a V$ in $\lambda^{\forall}_{rpc}$, then $[\![M]\!] \Downarrow_a [\![V]\!]$ in λ^{typed}_{rpc}.

Part III: A slicing compilation

After the monomorphization translation, $\lambda_{rpc}^{\forall}$ terms will become λ_{rpc}^{typed} terms.

Then the existing slicing compilation for λ_{rpc}^{typed} can be reused with no modification.

In summary, $\lambda_{rpc}^{\forall}$ is a complete RPC calculus because

- it supports polymorphic location programming, and
- it supports a slicing compilation into the client-server calculus.

Extension: A dynamic approach to $\lambda^{\forall}_{rpc}$

Instead of translating away all location abstractions and applications, the client-server calculus is extended to support them.

A new client-server calculus λ^{gen}_{cs} to support location abstractions and applications by dynamically passing locations in runtime

- $\lambda_{cs} : f(arg)$, req(f, arg), call(f, arg)
- $\lambda^{gen}_{cs} = \lambda_{cs} + gen(Loc, f, arg)$

The dynamic semantics of $gen(Loc', f, arg)$ at Loc: it behaves as one of the three procedure calls, chosen by Loc and Loc'.

Caller(Loc), Callee(Loc')	λ^{gen}_{cs}	Procedure call
$Loc = Loc'$	$f(arg)$	Local (Loc →Loc)
$Loc = \mathbf{c}$ and $Loc' = \mathbf{s}$	$req(f, arg)$	RPC (Loc→Loc')
$Loc = \mathbf{s}$ and $Loc' = \mathbf{c}$	$call(f, arg)$	RPC (Loc'→Loc)

Extension: A dynamic approach to $\lambda^{\forall}_{rpc}$(Cont.)

A slicing compilation: every application $L\ M$ in $\lambda^{\forall}_{rpc}$ is simply compiled into $gen(Loc', L, M)$ in λ^{gen}_{cs}

$$\frac{\Gamma \vdash_{Loc} L : A \xrightarrow{Loc'} B \qquad \Gamma \vdash_{Loc} M : A}{\Gamma \vdash_{Loc} L\ M : B}$$

Each instance of $gen(Loc', L, M)$ at Loc can be specialized before runtime into

- L(M) if $Loc = Loc'$
- req(L, M) if $Loc = \mathbf{c}$ and $Loc' = \mathbf{s}$
- call(L, M) if $Loc = \mathbf{s}$ and $Loc' = \mathbf{c}$

Otherwise ($(Loc = l$ or $Loc = l')$ and $Loc \neq Loc'$), unspecialized.

Regardless of the introduction of dynamic operations on locations, every remote procedure call identified statically will remain as RPC during evaluation.

Conclusion

The polymorphic RPC calculus that provides a parametric polymorphism over locations

- A static approach using the monomorphization translation
- A dynamic approach using the location-passing λ_{cs}^{gen}