SAFEDPI: using types to control mobile code

Matthew Hennessy, U of Sussex

- Background: controlling resources using types
- SAFEDPI: a higher-order distributed picalculus
- Process types
- Examples
- Behavioural equivalences

Supported by EU Global Computing projects Mikado/ Myths

Joint work with Julian Rathke, Yoshida Nobuko

Details in Sussex Technical Report

Extended abstract in Fossacs 2004

Background

Distributed processes: $l[\![P]\!] \mid (\mathsf{new}\, e : \mathsf{E})(k[\![Q]\!] \mid l[\![R]\!])$ Capability types ensure

- channels/resources are typesafe
- use of channels/resources policy driven

$$\textsc{serv}[\![(\mathsf{newloc}\, k : \mathsf{K})\, \mathsf{with}\, P \;\mathsf{in}\; \mathsf{xpt}_1!\langle k\rangle \mid \mathsf{xpt}_2!\langle k\rangle]\!]$$
$$\longrightarrow\; (\mathsf{new}\, k : \mathsf{K})\;\; \textsc{serv}[\![\mathsf{xpt}_1!\langle k\rangle \mid \mathsf{xpt}_2!\langle k\rangle]\!] \;\;\mid\;\; k[\![P]\!]$$

Client capabilities on location k depend on rights obtained via distribution channels xpt_1 and xpt_2

Goto Considered Harmful

Unrestricted migration:

$$k[\![\text{go } \textsc{serv}.\text{Nasty}]\!] \mid \textsc{serv}[\![S]\!] \longrightarrow \textsc{serv}[\![S \mid \text{Nasty}]\!]$$

With static typing:

- Nasty uses resources at SERV in type-safe fashion
- SERV has no control over immigration by Nasty

Objective: control migration and behaviour of incoming agents

Goto tamed in SAFEDPI

In SAFEDPI:

$$k[\![\mathsf{go}_p\ \textsc{serv}.\mathsf{Nasty}]\!] \mid \textsc{serv}[\![S]\!] \longrightarrow \textsc{serv}[\![S \mid p!\langle \mathsf{Nasty}\rangle]\!]$$

p: a port at site SERV - aka: higher-order channel

Nasty - a higher-order value - aka: thunked process

Nasty gains entrance if SERV provides access via port p

Using ports to control access

Server is interested:

$$k[\![\mathsf{go}_p\ \textsc{serv}.\mathsf{Nasty}]\!] \mid \textsc{serv}[\![S \mid p?(\xi)\ \mathsf{run}\ \xi]\!]$$

$$\longrightarrow \textsc{serv}[\![S \mid p?(\xi)\ \mathsf{run}\ \xi \mid p!\langle \mathsf{Nasty}\rangle]\!]$$

$$\longrightarrow \textsc{serv}[\![S \mid \mathsf{Nasty}]\!]$$

Server is not interested:

$$k[\![\mathsf{go}_p\ \textsc{serv}.\mathsf{Nasty}]\!] \mid \textsc{serv}[\![S]\!] \longrightarrow \textsc{serv}[\![S \mid p!\langle \mathsf{Nasty}\rangle]\!]$$

S without p

$$\equiv \textsc{serv}[\![S]\!]$$

Using typed ports to control behaviour

$$k[\![\mathsf{go}_p\ \textsc{serv}.\mathsf{Nasty}]\!] \mid \textsc{serv}[\![S \mid p?(\xi : \mathsf{G})\ \mathsf{run}\ \xi]\!]$$

$$\longrightarrow \textsc{serv}[\![S \mid p?(\xi : \mathsf{G})\ \mathsf{run}\ \xi \mid p!\langle \mathsf{Nasty} \rangle]\!]$$

$$\longrightarrow \textsc{serv}[\![S \mid \mathsf{Nasty}]\!]$$

Type G determines allowed behaviour of incoming Nasty

Idea: use process types from :

Assigning Types to Processes, Yoshida and Hennessy, LICS 2000

Process Types $\mathsf{pr}[\ldots]$

Process restricted to at most two channels:

$$\mathsf{pr}[\mathsf{info} : \mathsf{r}\langle\mathbf{str}\rangle@\mathsf{here},\ \mathsf{reply} : \mathsf{w}\langle\mathbf{str}\rangle@\textsc{cl}]$$

- read from local channel info
- write to channel reply at location CL

Process Types $\mathsf{pr}[\ldots]$

Process restricted to at most two channels:

$$\mathsf{pr}[\mathsf{info} : \mathsf{r}\langle \mathbf{str} \rangle_{@}\mathsf{here},\ \mathsf{reply} : \mathsf{w}\langle \mathbf{str} \rangle_{@}\textsc{cl}]$$

- read from local channel info
- write to channel reply at location $\textsc{cl}$

Process needs an entry port:

$$\mathsf{pr}[\mathsf{info} : \mathsf{r}\langle \mathbf{str} \rangle_{@}\mathsf{here},\ \mathsf{reply} : \mathsf{w}\langle \mathbf{str} \rangle_{@}\textsc{cl},\ \mathsf{in} : \mathsf{w}\langle \mathsf{thunk} \rangle_{@}\textsc{cl}]$$

Process Types $\mathsf{pr}[\ldots]$

Process restricted to at most two channels:

$$\mathsf{pr}[\mathsf{info} : \mathsf{r}\langle\mathbf{str}\rangle_{@}\mathsf{here},\ \mathsf{reply} : \mathsf{w}\langle\mathbf{str}\rangle_{@\mathrm{CL}}]$$

- read from local channel info
- write to channel reply at location CL

Process needs an entry port:

$$\mathsf{pr}[\mathsf{info} : \mathsf{r}\langle\mathbf{str}\rangle_{@}\mathsf{here},\ \mathsf{reply} : \mathsf{w}\langle\mathbf{str}\rangle_{@\mathrm{CL}},\ \mathsf{in} : \mathsf{w}\langle\mathsf{I}\rangle_{@\mathrm{CL}}]$$

$$\text{where}\ \ \mathsf{I} = \mathsf{th}[\mathsf{reply} : \mathsf{w}\langle\mathbf{str}\rangle_{@\mathrm{CL}}]_{@\mathrm{CL}}$$

$\mathtt{th}[\ldots]$ - thunk types

Syntax of SAFEDPI: Systems

$M, N \quad ::=$	*Systems*
$l[\![P]\!]$	Located Process
$M \mid N$	Composition
$(\text{new } e : \mathsf{E})\, M$	Name Creation
0	Termination
$u, v \quad ::=$	*Values*
$\lambda(\tilde{x} : \tilde{\mathsf{T}})P$	Scripts
$x, n, \ldots$	The usual: identifiers etc

Syntax of SAFEDPI: Processes

$P, Q ::=$	*Processes*
$u!\langle V \rangle$	Output
$u?(X : \mathsf{T})\, P$	Input
$\mathsf{go}_{u}\, v.P$	Migration
$\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$	Matching
$(\mathsf{newc}\, c : \mathsf{C})\ P$	Channel creation
$(\mathsf{newreg}\, n : \mathsf{N})\ P$	Global name creation
$(\mathsf{newloc}\, k : \mathsf{K})\ \mathsf{with}\ P\ \mathsf{in}\ Q$	Location creation
$P \mid Q$	Composition
$F\ (\tilde{v})$	Application
$* P$	Iteration
stop	Termination

Reduction Semantics

(R-COMM)	$k[\![c!\langle V\rangle]\!] \mid k[\![c?(X : \mathsf{T})\, P]\!] \longrightarrow k[\![P\{\!\!\{^{V}\!/\!_{X}\}\!\!\}]\!]$
(R-MOVE)	$k[\![\mathsf{go}_p\, l.F]\!] \longrightarrow l[\![p!\langle F\rangle]\!]$
(R-BETA)	$(\lambda\, (\tilde{x} : \tilde{\mathsf{T}}).\ P)(\tilde{v}) \longrightarrow P\{\!\!\{^{\tilde{v}}\!/\!_{\tilde{x}}\}\!\!\}$
(R-L.CREATE)	$k[\![(\mathsf{newloc}\, l : \mathsf{L})\, \mathsf{with}\, P\ \mathsf{in}\ Q]\!] \longrightarrow$ $(\mathsf{new}\, l : \mathsf{L})(k[\![Q]\!] \mid l[\![P]\!])$
...	......

Types in SAFEDPI

local channels	$\mathsf{C} ::= \mathsf{r}\langle \mathsf{T} \rangle \mid \mathsf{w}\langle \mathsf{T} \rangle \mid \mathsf{rw}\langle \mathsf{T}_r, \mathsf{T}_w \rangle$
locations	$\mathsf{L} ::= \mathsf{loc}[u_1 : \mathsf{C}_1, \ldots, u_n : \mathsf{C}_n]$
processes	$\pi ::= \mathbf{proc} \mid \mathsf{pr}[u_1 : \mathsf{C}_1 @ w_1, \ldots u_n : \mathsf{C} @ w_n]$
scripts	$\mathsf{S} ::= \mathsf{FDep}((\tilde{x} : \tilde{\mathsf{T}}) \rightarrow \pi)$
values	$\mathsf{T} ::= \mathsf{S} \mid \mathsf{C} \mid \mathsf{L} \mid$
	$\mathsf{TDep}(\tilde{\mathsf{T}})\,\mathsf{T} \mid \mathsf{EDep}(\tilde{\mathsf{T}})\,\mathsf{T}$

...

Dependent function types

$$\mathsf{FDep}(x : \mathsf{r}\langle \mathsf{T} \rangle \rightarrow \mathsf{pr}[x : \mathsf{r}\langle \mathsf{T} \rangle_{@}\mathsf{here},\ \mathsf{reply} : \mathsf{w}\langle \mathsf{T} \rangle_{@}k])$$

Script which is instantiated with a local channel; can only access

- that local channel in read mode
- channel reply at site k in write mode.

Dependent tuple types

$$\mathsf{TDep}(x:\mathsf{L})\,\mathsf{th}[\mathsf{info}:\mathsf{r}\langle\mathbf{str}\rangle_{@}\mathsf{here},\mathsf{reply}:\mathsf{w}\langle\mathbf{str}\rangle_{@}x]$$

Thunk, tupled with a location of type L; can access

- local channel info in read mode
- channel reply in write mode at provided location

thunk type $\mathsf{th}[\ldots\ldots]$ shorthand for script with no arguments $\mathsf{FDep}(() \rightarrow \mathsf{pr}[\ldots\ldots])$

run V shorthand for $V()$ where V is a thunk

Existential tuple types

$$\mathsf{EDep}(x : \mathsf{L})\, \mathsf{th}[\mathsf{info} : \mathsf{r}\langle \mathbf{str} \rangle_{@}\mathsf{here}, \mathsf{reply} : \mathsf{w}\langle \mathbf{str} \rangle_{@}x]$$

Thunk which can access

- local channel info in read mode
- channel reply in write mode at some location provided by client

Provided location can only be used as part of thunk

Server does not have independent use of provided location

Example

Client can only deliver news

Service: $s[\![\mathsf{req}?(\xi : \mathsf{S})\,\mathsf{run}\,\xi \;\mid\; *\,\mathsf{news}?(x)\;\textit{continue}]\!]$

Client: $\textsc{cl}[\![\mathsf{go}_{\mathsf{req}}\; s.\mathsf{news}!\langle\textit{scandal}\rangle]\!]$

Guardian type S: $\mathsf{th}[\mathsf{news} : \mathsf{w}\langle\mathbf{str}\rangle_{@}\mathsf{here}]$

Example

Client collects the news

Service: $s[\![\mathsf{req}?(\xi : \mathsf{S})\,\mathsf{run}\,\xi \;\mid\; *\,\mathsf{news}!\langle \textit{juicy}\rangle]\!]$

Client: $\mathsf{CL}[\![\mathsf{go}_{\mathsf{req}}\, s.\mathsf{news}?(x)\,\mathsf{go}_{\mathsf{in}}\,\mathsf{CL}.\,\mathsf{reply}!\langle x\rangle$

$\mid \mathsf{in}?(\xi : R)\,\mathsf{run}\,\xi \mid \mathsf{reply}?(y)\;\textit{continue}]\!]$

Guardians:

$\mathsf{S} : \mathsf{th}[\mathsf{news} : \mathsf{r}\langle \mathbf{str}\rangle_{@}\mathsf{here},\ \mathsf{in} : \mathsf{w}\langle \mathsf{thunk}\rangle_{@}\mathsf{CL}]$

$\mathsf{R} : \mathsf{thunk}$

Example

Client collects the news

Service: $s[\![\text{req}?(\xi : \mathsf{S})\,\text{run}\,\xi \;\mid\; *\,\text{news}!\langle \textit{juicy} \rangle]\!]$

Client: $\textsc{cl}[\![\text{go}_{\text{req}}\, s.\text{news}?(x)\,\text{go}_{\text{in}}\,\textsc{cl}.\,\text{reply}!\langle x \rangle$

$\mid \text{in}?(\xi : R)\,\text{run}\,\xi \mid \text{reply}?(y)\; \textit{continue}]\!]$

Guardians for client protection:

$\mathsf{S} : \text{th}[\text{news} : \text{r}\langle \mathbf{str} \rangle_{@}\text{here},\ \text{in} : \text{w}\langle \mathsf{R} \rangle_{@}\textsc{cl}]$

$\mathsf{R} : \text{th}[\text{reply} : \text{w}\langle \mathbf{str} \rangle_{@}\text{here}]$

Anonymous servers - dependent tuple types

server does not know clients name

Service: $s[\![\mathsf{req}?(\xi \text{ with } x : \mathsf{S})\,\mathsf{run}\,\xi \mid *\,\mathsf{news}!\langle \textit{juicy} \rangle]\!]$

Client: $\mathsf{CL}[\![\mathsf{go}_{\mathsf{req}}\ s.\mathsf{news}?(x)\,\mathsf{go}_{\mathsf{in}}\ \mathsf{CL}.\,\mathsf{reply}!\langle x \rangle \text{ with } \mathsf{CL}]\!]$

$\mid \mathsf{in}?(\xi : R)\,\mathsf{run}\,\xi \mid \mathsf{reply}?(y)$ *continue*

Guardians:

$\mathsf{S} : \mathsf{TDep}(x : \mathsf{In})\ \ \mathsf{th}[\mathsf{news} : \mathsf{r}\langle \mathbf{str} \rangle @\mathsf{here},\ \mathsf{in} : \mathsf{w}\langle \mathsf{thunk} \rangle @x]$

$\mathsf{R} : \mathsf{thunk}$

$\mathsf{In} : \mathsf{loc}[\mathsf{in} : \mathsf{w}\langle \mathsf{thunk} \rangle]$

Anonymous servers - protecting clients

Service: $s[\![\mathsf{req}?(\xi \text{ with } x, y, z : \mathsf{S})\, \mathsf{run}\, \xi \mid *\, \mathsf{news}!\langle \textit{juicy} \rangle]\!]$

Client: $\textsc{cl}[\![(\mathsf{newc\ reply})\ (\mathsf{newc\ in} : \mathsf{rw}\langle \mathsf{R} \rangle)$

$\mathsf{go}_{\mathsf{req}}\, s.\mathsf{news}?(x)\, \mathsf{go}_{\mathsf{in}}\, \textsc{cl}.\, \mathsf{reply}!\langle x \rangle \text{ with } (\textsc{cl}, \mathsf{reply}, \mathsf{in})$

$\mid \mathsf{in}?(\xi : R)\, \mathsf{run}\, \xi \mid \mathsf{reply}?(y)\ \textit{continue}]\!]$

Guardians:

$$\mathsf{S} : \begin{array}{l} \mathsf{TDep}(x : \mathsf{loc},\ y : \mathsf{w}\langle \mathbf{str} \rangle @x,\ z : \mathsf{w}\langle \mathsf{In}_{x,y} \rangle @x) \\ \qquad \mathsf{th}[\mathsf{news} : \mathsf{r}\langle \mathbf{str} \rangle @\mathsf{here},\ z : \mathsf{w}\langle \mathsf{In}_{x,y} \rangle @x] \end{array}$$

$$\mathsf{R} : \mathsf{w}\langle \mathsf{th}[\mathsf{reply} : \mathsf{w}\langle \mathbf{str} \rangle] \rangle$$

$\mathsf{In}_{x,y} : \mathsf{th}[y : \mathsf{w}\langle \mathbf{str} \rangle @x]$

Nasty servers

Service: $s[\![\mathsf{req}?(\xi \text{ with } x, y, z : \mathsf{S})\, \mathsf{run}\, \xi \;|\; \mathsf{go}_z\, x.y!\langle \textit{rubbish} \rangle]\!]$

Client: $\mathsf{CL}[\![\ldots\ldots \mathsf{go}_{\mathsf{req}}\, s.\ldots \text{ with } (\mathsf{CL}, \mathsf{reply}, \mathsf{in})]\!]$

Server

ignores incoming script

lifts return address from it

misleads client

Existential types

protecting client information from nasty servers

Service: $s[\![\mathsf{req}?(\xi : \mathsf{S}_e)\,\mathsf{run}\,\xi \mid *\,\mathsf{news}!\langle\textit{juicy}\rangle]\!]$

Client: $\mathsf{CL}[\![\ldots\ldots\mathsf{go}_{\mathsf{req}}\, s.\ldots\,\mathsf{with}\,(\mathsf{CL}, \mathsf{reply}, \mathsf{in})]\!]$

$$\mathsf{S}_e : \quad \mathsf{EDep}(x : \mathsf{loc},\ y : \mathsf{w}\langle\mathbf{str}\rangle@x,\ z : \mathsf{w}\langle\mathsf{In}_{x,y}\rangle@x)$$
$$\mathsf{th}[\mathsf{news} : \mathsf{r}\langle\mathbf{str}\rangle@\mathsf{here},\ z : \mathsf{w}\langle\mathsf{In}_{x,y}\rangle@x,\ y : \mathsf{w}\langle\mathbf{str}\rangle@x]$$

Server does not gain access to $(\mathsf{CL}, \mathsf{reply}, \mathsf{in})$

Service: $s[\![\mathsf{req}?(\xi : \mathsf{S}_e)\,\mathsf{run}\,\xi \mid \mathsf{go}_z\, x.y!\langle\textit{rubbish}\rangle]\!]$

not well-typed

Behavioural Equivalences

Two problems:

- Capability types: observers may not have full knowledge of processes
- Higher-order language

Behavioural Equivalences

Two problems:

- Capability types: observers may not have full knowledge of processes

  Use *typed bisimulations* from: Towards a . . . Mobility . . . , by Hennessy, Merro, Rathke, in Fossacs 2003

- Higher-order language

  Target *higher-order* bisimulations for the moment

Typed contextual equivalence

$$\mathcal{I} \models M \approx_{cxt} N$$

M and N can not be distinguished by any observer typeable by $\mathcal{I}$

$\mathcal{I}$ is current observers knowledge of resources/capabilities of $M,\ N$.

Typed contextual equivalence

$$\mathcal{I} \models M \approx_{cxt} N$$

M and N can not be distinguished by any observer typeable by $\mathcal{I}$

$\mathcal{I}$ is current observers knowledge of resources/capabilities of M, N.

$$\mathcal{I} \models k[\![\mathsf{xpt}!\langle \mathsf{req}!\langle \mathsf{news}\rangle\rangle]\!] \approx_{cxt} k[\![\mathsf{xpt}!\langle \mathsf{stop}\rangle]\!]$$

Typed contextual equivalence

$$\mathcal{I} \models\ M \approx_{cxt} N$$

M and N can not be distinguished by any observer typeable by $\mathcal{I}$

$\mathcal{I}$ is current observers knowledge of resources/capabilities of $M,\ N$.

$$\mathcal{I} \models\ k[\![\mathsf{xpt!}\langle \mathsf{req!}\langle \mathsf{news}\rangle\rangle]\!] \approx_{cxt} k[\![\mathsf{xpt!}\langle \mathsf{stop}\rangle]\!]$$

provided req at k not known in $\mathcal{I}$

Typed higher-order actions

- Internal actions: $\mathcal{I} \triangleright M \xrightarrow{\tau} \mathcal{I} \triangleright N$
- Input actions: $\mathcal{I} \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathsf{E}})k.c?V} \mathcal{I}' \triangleright N$
- Output Actions: $\mathcal{I} \triangleright M \xrightarrow{(\tilde{n})k.c!G} \mathcal{I}' \triangleright N$

Typed higher-order actions

- Internal actions: $\mathcal{I} \triangleright M \xrightarrow{\tau} \mathcal{I} \triangleright N$
- Input actions: $\mathcal{I} \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathsf{E}})k.c?V} \mathcal{I}' \triangleright N$

$$\mathcal{I} \triangleright k[\![\mathsf{req}?(\xi)\,\mathsf{run}\,\xi \mid S]\!] \xrightarrow{k.\mathsf{req}?V} \mathcal{I} \triangleright k[\![\mathsf{run}\,V \mid S]\!]$$

- Output Actions: $\mathcal{I} \triangleright M \xrightarrow{(\tilde{n})k.c!G} \mathcal{I}' \triangleright N$

$$\mathcal{I} \triangleright k[\![\mathsf{req}!\langle V\rangle \mid S]\!] \xrightarrow{k.\mathsf{req}!G} \mathcal{I}, \ldots \triangleright k[\![GV \mid S]\!]$$

Typed higher-order actions

- Internal actions: $\mathcal{I} \triangleright M \xrightarrow{\tau} \mathcal{I} \triangleright N$
- Input actions: $\mathcal{I} \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathsf{E}})k.c?V} \mathcal{I}' \triangleright N$

$$\mathcal{I} \triangleright k[\![\mathsf{req}?(\xi)\,\mathsf{run}\,\xi \mid S]\!] \xrightarrow{k.\mathsf{req}?V} \mathcal{I} \triangleright k[\![\mathsf{run}\,V \mid S]\!]$$

provided $\mathcal{I}$ knows req at k, ...

- Output Actions: $\mathcal{I} \triangleright M \xrightarrow{(\tilde{n})k.c!G} \mathcal{I}' \triangleright N$

$$\mathcal{I} \triangleright k[\![\mathsf{req}!\langle V\rangle \mid S]\!] \xrightarrow{k.\mathsf{req}!G} \mathcal{I}, \ldots \triangleright k[\![GV \mid S]\!]$$

provided $\mathcal{I}$ knows req at k, $\mathcal{I}$ types G appropriately, ...

Typed higher-order actions

- Internal actions: $\mathcal{I} \triangleright M \xrightarrow{\tau} \mathcal{I} \triangleright N$
- Input actions: $\mathcal{I} \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathsf{E}})k.c?V} \mathcal{I}' \triangleright N$

$$\mathcal{I} \triangleright k[\![\mathsf{req}?(\xi)\,\mathsf{run}\,\xi \mid S]\!] \xrightarrow{k.\mathsf{req}?V} \mathcal{I} \triangleright k[\![\mathsf{run}\,V \mid S]\!]$$

provided $\mathcal{I}$ knows **req** at k, . . . and $\mathcal{I}$ has migration rights to k

- Output Actions: $\mathcal{I} \triangleright M \xrightarrow{(\tilde{n})k.c!G} \mathcal{I}' \triangleright N$

$$\mathcal{I} \triangleright k[\![\mathsf{req}!\langle V \rangle \mid S]\!] \xrightarrow{k.\mathsf{req}!G} \mathcal{I}, \ldots \triangleright k[\![GV \mid S]\!]$$

provided $\mathcal{I}$ knows **req** at k, $\mathcal{I}$ types G appropriately, . . . and $\mathcal{I}$ has migration rights to k

Higher-order goto actions

$$\mathcal{I} \triangleright M \xrightarrow{\mathsf{go}_p\, k.V} \mathcal{I}, \triangleright M \mid k[\![p!\langle V \rangle]\!]$$

Higher-order goto actions

$$\mathcal{I} \triangleright M \xrightarrow{\mathsf{go}_p\, k.V} \mathcal{I}, \triangleright M \mid k[\![p!\langle V \rangle]\!]$$

provided $\mathcal{I}$ knows about p at k and V at appropriate type

Higher-order goto actions

$$\mathcal{I} \triangleright M \xrightarrow{\mathsf{go}_p\, k.V} \mathcal{I}, \triangleright M \mid k[\![p!\langle V \rangle]\!]$$

provided $\mathcal{I}$ knows about p at k and V at appropriate type

even if $\mathcal{I}$ has no migration rights to k

$\mathcal{T}$ Contextuality

$\mathcal{T}$: locations to which $\mathcal{I}$ has migration rights

bisimulation equivalence is contextual

- $\mathcal{I} \models M \approx_{bis}{}^{\mathcal{T}} N$ and
- $\mathcal{I} \vdash k[\![O]\!]$
- k in $\mathcal{T}$

implies

$$\mathcal{I} \models M \mid k[\![O]\!] \approx_{bis}{}^{\mathcal{T}} M \mid k[\![O]\!]$$

Example: Observer $k[\![\mathsf{req}?(\xi)\ P]\!]$ captured by action $k.\mathsf{req}!\langle\lambda\ \xi.\ P\rangle$

Full Abstraction

$\mathcal{I} \models M \approx_{bis}{}^{\mathcal{T}} N$ if and only if $\mathcal{I} \models M \approx_{cxt}{}^{\mathcal{T}} N$