Static vs Dynamic Typing for Access Control in Pi-Calculus

Michele Bugliesi
Damiano Macedonio
Sabina Rossi

Department of Informatics
University of Venice

Doha, December 11, 2007

Asynchronous Pi-calculus

Channels $a, b, \ldots, n, m, \ldots$

Processes $P, Q, \ldots$

$\mathbf{0}$	Inaction	$(\nu n{:}T)P$	Restriction
$\overline{a}\langle\tilde{v}\rangle$	Output	$[u{=}v]P; Q$	Matching
$a(\tilde{x}).P$	Input	$!\,P$	Replication
$P \mid Q$	Composition		

(OUTPUT) $\quad \overline{a}\langle v\rangle \xrightarrow{\overline{a}\langle v\rangle} \mathbf{0}$

(INPUT) $\quad a(x).P \xrightarrow{a(v)} P\{v/x\}$

(COMMUNICATION) $$\frac{P \xrightarrow{\overline{a}\langle b\rangle} P' \qquad Q \xrightarrow{a(b)} Q'}{P \mid Q \longrightarrow P' \mid Q'}$$

Static Typing in APi (Hennessy-Rathke 2004)

Types define access rights on channels.

$$T, S ::= \mathsf{rw}\langle\tilde{S};\tilde{T}\rangle \mid \mathsf{r}\langle\tilde{S}\rangle \mid \mathsf{w}\langle\tilde{T}\rangle \mid \top \mid X \mid \mu X.T$$

Subtyping: Higher types grant fewer access rights ($<:$)

Intention:

- Control interaction with types (based on access rights).
- Control propagation of access rights.

Formalisation: Typed Labelled Transitions

$$\mathcal{I} \triangleright P \xrightarrow{\alpha} \mathcal{I}' \triangleright P'$$

- $\mathcal{I}$ constraints the behaviour of contexts for P.
- P is well typed in a (more precise) $\Gamma <: \mathcal{I}$.

Access Rights Propagation

Interaction determines how access rights are propagated

$$(\textsc{aPi-output}) \quad \frac{\mathcal{I}^{\mathsf{r}}(a)\downarrow}{\mathcal{I} \triangleright \overline{a}\langle v\rangle \xrightarrow{\overline{a}\langle v\rangle} \mathcal{I}, v : \mathcal{I}^{\mathsf{r}}(a) \triangleright \mathbf{0}}$$

$$(\textsc{aPi-input}) \quad \frac{\mathcal{I}^{\mathsf{w}}(a)\downarrow \qquad \mathcal{I} \vdash v : \mathcal{I}^{\mathsf{w}}(a)}{\mathcal{I} \triangleright a(x).P \xrightarrow{a(v)} \mathcal{I} \triangleright P\{v/x\}}$$

Behavioural Equivalence:
Based on this formalisation $\mathcal{I} \models P \approx Q$

$\mathcal{I}$ matters! For instance $a : \mathsf{w}\langle\rangle \models \overline{a}\langle\rangle \approx \mathbf{0}$

Typed Processes in Untyped Contexts

In establishing $\mathcal{I} \models P \approx Q$ we assume

- contexts have knowledge $\mathcal{I}$
- contexts are well typed (statically)

Types are very informative $\Rightarrow$ strong control on behaviour.

If we drop well typing, then everything falls apart.

Typed Processes in Untyped Contexts

In establishing $\mathcal{I} \models P \approx Q$ we assume

- contexts have knowledge $\mathcal{I}$
- contexts are well typed (statically)

Types are very informative $\Rightarrow$ strong control on behaviour.

If we drop well typing, then everything falls apart.

Idea: Use simpler types. Only Top-Level capabilities.

- Well typing provides looser control
- Easier to enforce with type coercion: $\overline{a}\langle v@A\rangle$ and $a(x@A)$
- A completely different interaction.
- Processes are still in control with much simpler assumptions on contexts.

Dynamic Typing in $\mathrm{API@}$ (Bugliesi-Giunti 2005)

Processes

$$P, Q \quad ::= \quad \mathbf{0} \mid \overline{a}\langle \tilde{v}@\tilde{A}\rangle \mid a(\tilde{x}@\tilde{A}).P \mid P|Q \mid (\nu n{:}T)P \mid [u{=}v]P; Q \mid !P$$

Types

$$A, B \quad ::= \quad \mathsf{rw} \mid \mathsf{r} \mid \mathsf{w} \mid \top$$

Still statically typed

$$\frac{\Gamma(a) <: \mathsf{w} \quad \Gamma \vdash v : A}{\Gamma \vdash \overline{a}\langle v@A\rangle : \checkmark} \qquad \frac{\Gamma(a) <: \mathsf{r} \quad \Gamma, x : A \vdash P : \checkmark}{\Gamma \vdash a(x@A).P : \checkmark}$$

Plus dynamically typed synchronisation

$$\frac{P \xrightarrow{\overline{a}\langle b@A\rangle} P' \quad Q \xrightarrow{a(b@B)} Q' \quad A <: B}{P \mid Q \longrightarrow P' \mid Q'}$$

Access Rights Propagation in API@

The sender determines how access rights are propagated

$$(\text{API@-OUTPUT}) \quad \frac{\mathcal{I}^{\mathsf{r}}(a)\downarrow \qquad A <: B}{\mathcal{I} \triangleright \overline{a}\langle v@A\rangle \xrightarrow{\overline{a}\langle v@B\rangle} \mathcal{I}, v : B \triangleright \mathbf{0}}$$

$$(\text{API-INPUT}) \quad \frac{\mathcal{I}^{\mathsf{w}}(a)\downarrow \qquad \mathcal{I} \vdash v : B \qquad B <: A}{\mathcal{I} \triangleright a(x@A).P \xrightarrow{a(v@B)} \mathcal{I} \triangleright P\{v/x\}}$$

Behavioural Equivalence:
Based on this formalisation $\mathcal{I} \models P \approx^{@} Q$

Implementation

- API@ serves the purpose of studying the behaviour of typed processes in un-typed contexts
- In (POPL'07) Bugliesi-Giunti defined an implementation of API@ as an encoding $[\![\ ]\!] : \text{API@} \longrightarrow \textit{Applied}\pi$
- Main result: $\mathcal{I} \models P \approx^{@} Q$ if and only if $[\![\mathcal{I}]\!] \models [\![P]\!] \approx^{A\pi} [\![Q]\!]$
  Idea:
  - Translate types of API@ into crypto-keys that give access to communication channels.
  - Use cryptography to control propagation in a way that mimics the propagation in API@.
  - Equivalences in *Applied*π are established in environments with knowledge of certain keys (those that correspond to the types in API@).

Dynamic Typing vs Static Typing

Which is the relationship between API and API@?

- Current results:

$$\llbracket\ \rrbracket_1 \;:\; \text{API} \longrightarrow \text{API@} \quad \text{sound and divergence free}$$

$$\llbracket \overline{a}\langle v\rangle \rrbracket_\Gamma \stackrel{def}{=} \overline{a}\langle v@|\Gamma^w(a)|\rangle$$

$$\llbracket a(x).P \rrbracket_\Gamma \stackrel{def}{=} a(x@|\Gamma^r(a)|).\llbracket P \rrbracket_{\Gamma,x:\Gamma^r(a)}$$

$$\llbracket\ \rrbracket_2 \;:\; \text{API@} \longrightarrow \text{API} \quad \text{sound and complete, but divergent}$$

- Under "appropriate" hypothesis we can also show that
  - No encoding API $\rightarrow$ API@ can be sound and complete.
  - No encoding API@ $\rightarrow$ API can be divergence free.

... but the "appropriate" hypothesis are currently too strong to make these negative results interesting.

The Encoding

$$[\![\ ]\!] : (\text{monadic})\textsc{API@} \longrightarrow (\text{poliadic})\textsc{API}$$

The encoding is defined in terms of two related, but independent mappings.

- The encoding of processes maps typing judgments in API@ to processes of API.
- The encoding of type environments maps capabilities (types) of the observing API@ contexts into the corresponding capabilities of API contexts.

First Attempt

- We associate every API@-name n with a tuple of API-names $\underline{n} = (n_{rw}, n_r, n_w, n_\top)$
- A synchronisation on n at type B in API@ can be seen as a synchronisation on n_B in API.
- Being more precise: inputs (outputs) at type B can synchronise at a type which is lower (upper) than B.
- Then we may think at the following encoding:

$$\begin{array}{lrcl} \text{Output} & [\![\, \overline{n}\langle a@A\rangle \,]\!] & \stackrel{def}{=} & \overline{n_A}\langle \underline{n} \rangle \\ \text{Input} & [\![\, n(x@A).P \,]\!] & \stackrel{def}{=} & \sum_{B<:A} n_B(\underline{x}).[\![\, P \,]\!] \end{array}$$

- Now we only have to compose this idea with the encoding of input guarded choice in the asynchronous Pi-calculus.

Encoding Guarded Choice (Nestmann-Pierce 2000)

$\sum_{B<:A} n_B(\underline{x}).[\![P]\!]$

- Each branch of the sum is represented by a parallel branch.
- All the parallel branches run a mutual exclusion protocol, installing a local lock.
- All the parallel branches try to consume an output. They test the lock after reading a message from the environment.
- Each branch can black out and return to its initial state after it has taken the lock, just by re-sending the message.
- Just one branch will proceed with its continuation and thereby commit the input.
- Every other branch will then be forced to re-send the message and abort its continuation.

Encoding Guarded Choice (Nestmann-Pierce 2000)

$\sum_{B<:A} n_B(\underline{x}).[\![P]\!]$

- Each branch of the sum is represented by a parallel branch.
- All the parallel branches run a mutual exclusion protocol, installing a local lock.
- All the parallel branches try to consume an output. They test the lock after reading a message from the environment.
- Each branch can black out and return to its initial state after it has taken the lock, just by re-sending the message.
- Just one branch will proceed with its continuation and thereby commit the input.
- Every other branch will then be forced to re-send the message and abort its continuation.

It looks good, but...

Encoding Guarded Choice (Nestmann-Pierce 2000)

$\sum_{B<:A} n_B(\underline{x}).[\![P]\!]$

- Each branch of the sum is represented by a parallel branch.
- All the parallel branches run a mutual exclusion protocol, installing a local lock.
- All the parallel branches try to consume an output. They test the lock after reading a message from the environment.
- Each branch can black out and return to its initial state after it has taken the lock, just by re-sending the message.
- Just one branch will proceed with its continuation and thereby commit the input.
- Every other branch will then be forced to re-send the message and abort its continuation.

It looks good, but... it does not work!
All the readers must also be given the w-rights on the channel.

Fixing the Typing Problem

- Every APi@ channel n is associated to a process CHAN(n) that mediates between inputs and outputs.
- Each exchange on n in APi@ corresponds to running two separate protocols.
  - Input. A process willing to input on n at type A sends a read request (in the form of a private name) on the name $n_{\mathsf{r}@A}$.
  - Output. A process willing to output on n at type A sends its output on $n_{\mathsf{w}@A}$.
- Collectively, each name n from APi@ is thus translated into the 8-tuple $\underline{n} = (n_{\mathbb{R}}, n_{\mathbb{W}})$, where

$$
\begin{aligned}
n_{\mathbb{R}} &= n_{\mathsf{r}@\mathsf{rw}}, n_{\mathsf{r}@\mathsf{r}}, n_{\mathsf{r}@\mathsf{w}}, n_{\mathsf{r}@\top} \\
n_{\mathbb{W}} &= n_{\mathsf{w}@\mathsf{rw}}, n_{\mathsf{w}@\mathsf{r}}, n_{\mathsf{w}@\mathsf{w}}, n_{\mathsf{w}@\top}
\end{aligned}
$$

Encoding of Processes

Channel Servers

$$\mathrm{CHAN}(n) \stackrel{def}{=} \prod_{A \in \{\mathrm{rw},\mathrm{r},\mathrm{w},\top\}} !\, n_{\mathrm{r}@A}(h).\mathrm{CHOOSE}(n, A, h)$$

$$\mathrm{CHOOSE}(n, A, h) \stackrel{def}{=} (\boldsymbol{\nu} l : \mathrm{rw}\langle\top\rangle)\left(\bar{l}\langle \mathtt{t}\rangle \mid \prod_{B <: A} !\,\mathrm{READ}_l \left\langle n_{\mathrm{w}@B}(\underline{z}).\overline{h}\langle \underline{z}\rangle\right\rangle\right)$$

Clients

$$\{\!|\, \mathbf{0} \,|\!\} \stackrel{def}{=} \mathbf{0}$$

$$\{\!|\, \overline{u}\langle v@A\rangle \,|\!\} \stackrel{def}{=} \overline{u_{\mathrm{w}@A}}\langle \underline{v}\rangle$$

$$\{\!|\, u(x@A).P \,|\!\} \stackrel{def}{=} (\boldsymbol{\nu} h : \mathrm{rw}\langle \mathbb{T}_A\rangle)\,(\overline{u_{\mathrm{r}@A}}\langle h\rangle \mid h(\underline{x}).\{\!|\, P \,|\!\}\,)$$

$$\{\!|\, P \mid Q \,|\!\} \stackrel{def}{=} \{\!|\, P \,|\!\} \mid \{\!|\, Q \,|\!\}$$

$$\{\!|\, (\boldsymbol{\nu} a : A)P \,|\!\} \stackrel{def}{=} (\boldsymbol{\nu} \underline{a} : \mathbb{S})\,(\{\!|\, P \,|\!\} \mid \mathrm{CHAN}(a))$$

$$\{\!|\, [u = v]P; Q \,|\!\} \stackrel{def}{=} [u_{\mathrm{r}@\mathrm{r}} = v_{\mathrm{r}@\mathrm{r}}]\{\!|\, P \,|\!\}; \{\!|\, Q \,|\!\}$$

$$\{\!|\, !P \,|\!\} \stackrel{def}{=} !\,\{\!|\, P \,|\!\}$$

Complete Systems

$$\{\!|P|\!\}_\Gamma \stackrel{def}{=} \{\!|\, P \,|\!\} \mid \prod_{a \in dom(\Gamma)} \mathrm{CHAN}(a)$$

Encoding of Types

- A read capability on n in API@ corresponds in API to a write capability on all the names in $n_{\mathbb{R}}$.
- A write capability on n in API@ corresponds to a write capability on the names $n_{\mathbb{W}}$.
- With each type A in API@ we associate a corresponding tuple of types $\mathbb{T}_A$

Client Types

$$\mathbb{T}_{\mathrm{rw}} \stackrel{def}{=} (\mathbb{R}, \mathbb{W}) \quad \mathbb{T}_{\mathrm{r}} \stackrel{def}{=} (\mathbb{R}, \top) \quad \mathbb{T}_{\mathrm{w}} \stackrel{def}{=} (\top, \mathbb{W}) \quad \mathbb{T}_{\top} \stackrel{def}{=} (\top, \top)$$

$$\mathbb{R} \stackrel{def}{=} (\boldsymbol{T}_{\mathrm{r@rw}}, \boldsymbol{T}_{\mathrm{r@r}}, \boldsymbol{T}_{\mathrm{r@rw}}, \boldsymbol{T}_{\mathrm{r@}\top}) \quad \mathbb{W} \stackrel{def}{=} (\boldsymbol{T}_{\mathrm{w@rw}}, \boldsymbol{T}_{\mathrm{w@r}}, \boldsymbol{T}_{\mathrm{w@rw}}, \boldsymbol{T}_{\mathrm{w@}\top})$$

$$\begin{array}{lcl@{\qquad}lcl}
\boldsymbol{T}_{\mathrm{r@rw}} & \stackrel{def}{=} & \mathrm{w}\langle \mathrm{w}\langle \mathbb{R}, \mathbb{W}\rangle\rangle & \boldsymbol{T}_{\mathrm{w@rw}} & \stackrel{def}{=} & \mathrm{w}\langle \mathbb{R}, \mathbb{W}\rangle \\
\boldsymbol{T}_{\mathrm{r@r}} & \stackrel{def}{=} & \mathrm{w}\langle \mathrm{w}\langle \mathbb{R}, \top\rangle\rangle & \boldsymbol{T}_{\mathrm{w@r}} & \stackrel{def}{=} & \mathrm{w}\langle \mathbb{R}, \top\rangle \\
\boldsymbol{T}_{\mathrm{r@w}} & \stackrel{def}{=} & \mathrm{w}\langle \mathrm{w}\langle \top, \mathbb{W}\rangle\rangle & \boldsymbol{T}_{\mathrm{w@w}} & \stackrel{def}{=} & \mathrm{w}\langle \top, \mathbb{W}\rangle \\
\boldsymbol{T}_{\mathrm{r@}\top} & \stackrel{def}{=} & \mathrm{w}\langle \mathrm{w}\langle \top, \top\rangle\rangle & \boldsymbol{T}_{\mathrm{w@}\top} & \stackrel{def}{=} & \mathrm{w}\langle \top, \top\rangle
\end{array}$$

Type Environments

$$\{\!|\, \emptyset \,|\!\} \stackrel{def}{=} \mathtt{t} : \top, \mathtt{f} : \top \qquad \{\!|\, \Gamma, v : A \,|\!\} \stackrel{def}{=} \{\!|\, \Gamma \,|\!\}, (\underline{v}) : \mathbb{T}_A$$

Soundness

Theorem (Typing and Subtyping Preservation)

For all types A, B *in* API@*,* $A <: B$ *implies* $\mathbb{T}_A <: \mathbb{T}_B$ *in* API*.*
Furthermore, whenever $\Gamma \vdash P$ *in* API@*,*
then there exists $\Gamma' <: \{\!|\,\Gamma\,|\!\}$ *such that* $\Gamma' \vdash \{\!|P|\!\}_\Gamma$ *in* API*.*

Theorem (Soundness)

Let $\Gamma <: \mathcal{I}$ *and* $\Gamma' <: \mathcal{I}$ *be two type environments such that* $\Gamma \vdash P$ *and* $\Gamma' \vdash Q$ *in* API@*.*
Then $\{\!|\,\mathcal{I}\,|\!\} \models \{\!|P|\!\}_\Gamma \approx \{\!|Q|\!\}_{\Gamma'}$ *implies* $\mathcal{I} \models P \approx^{@} Q$*.*

Soundness

Theorem (Typing and Subtyping Preservation)

For all types A, B *in* API@, $A <: B$ *implies* $\mathbb{T}_A <: \mathbb{T}_B$ *in* API.
Furthermore, whenever $\Gamma \vdash P$ *in* API@,
then there exists $\Gamma' <: \{\!|\, \Gamma \,|\!\}$ *such that* $\Gamma' \vdash \{\!|P|\!\}_\Gamma$ *in* API.

Theorem (Soundness)

Let $\Gamma <: \mathcal{I}$ *and* $\Gamma' <: \mathcal{I}$ *be two type environments such that* $\Gamma \vdash P$ *and* $\Gamma' \vdash Q$ *in* API@.
Then $\{\!|\, \mathcal{I} \,|\!\} \models \{\!|P|\!\}_\Gamma \approx \{\!|Q|\!\}_{\Gamma'}$ *implies* $\mathcal{I} \models P \approx^{@} Q$.

The converse direction does not hold!

Soundness

Theorem (Typing and Subtyping Preservation)

For all types A, B in API@*,* $A <: B$ *implies* $\mathbb{T}_A <: \mathbb{T}_B$ *in* API*.*
Furthermore, whenever $\Gamma \vdash P$ *in* API@*,*
then there exists $\Gamma' <: \{\!|\, \Gamma \,|\!\}$ *such that* $\Gamma' \vdash \{\!|P|\!\}_\Gamma$ *in* API*.*

Theorem (Soundness)

Let $\Gamma <: \mathcal{I}$ *and* $\Gamma' <: \mathcal{I}$ *be two type environments such that* $\Gamma \vdash P$ *and* $\Gamma' \vdash Q$ *in* API@*.*
Then $\{\!|\, \mathcal{I} \,|\!\} \models \{\!|P|\!\}_\Gamma \approx \{\!|Q|\!\}_{\Gamma'}$ *implies* $\mathcal{I} \models P \approx^{@} Q$*.*

The converse direction does not hold!

The properties of the communication protocols are based on certain invariants that are verified by the names and the channel servers allocated by the encoding, but may fail for the names created dynamically by the context.

How to Recover Full Abstraction (Just an Idea)

- We need to protect the clients generated by the encoding from direct interactions on context generated names.

How to Recover Full Abstraction (Just an Idea)

- We need to protect the clients generated by the encoding from direct interactions on context generated names.
- We relies on a PROXY service to filter the interactions between channel servers, clients and the context.

How to Recover Full Abstraction (Just an Idea)

- We need to protect the clients generated by the encoding from direct interactions on context generated names.
- We relies on a PROXY service to filter the interactions between channel servers, clients and the context.
- The PROXY introduces a separation between client names and the corresponding proxy names.

How to Recover Full Abstraction (Just an Idea)

- We need to protect the clients generated by the encoding from direct interactions on context generated names.
- We relies on a Proxy service to filter the interactions between channel servers, clients and the context.
- The Proxy introduces a separation between client names and the corresponding proxy names.
- The Proxy maintains an association map between client and server names.

How to Recover Full Abstraction (Just an Idea)

- We need to protect the clients generated by the encoding from direct interactions on context generated names.
- We relies on a PROXY service to filter the interactions between channel servers, clients and the context.
- The PROXY introduces a separation between client names and the corresponding proxy names.
- The PROXY maintains an association map between client and server names.
- Read and write protocols:
they follow the same rationale as in the previous encoding, but a client must obtain the access to the system channel with a request to PROXY before starting the protocols.

How to Recover Full Abstraction (Just an Idea)

- We need to protect the clients generated by the encoding from direct interactions on context generated names.
- We relies on a PROXY service to filter the interactions between channel servers, clients and the context.
- The PROXY introduces a separation between client names and the corresponding proxy names.
- The PROXY maintains an association map between client and server names.
- Read and write protocols:
they follow the same rationale as in the previous encoding, but a client must obtain the access to the system channel with a request to PROXY before starting the protocols.
- Interaction between clients and PROXY:
the client presents a name to the PROXY,
the PROXY replies with the corresponding server name.

Conclusion

- We have given a fully abstract encoding of (monadic)API@ into (poliadic)API with recursive types.
- The same technique would work for the polyadic calculus, as long as we can count on a finite bound on the maximal arity.
- We could do without recursive types in API by assuming a finite bound on the number of cascading re-transmission via other names.
- The encoding shows how the dynamically typed synchronization of API@ may be simulated by a combination of untyped synchronizations on suitably designed channels, and it allows us to identify precisely the subclass of the static types of API that correspond to the dynamic types of API@.