Session Types as Generic Process Types

Simon J. Gay[1] Nils Gesbert[1] António Ravara[2,3]

[1]Department of Computing Science, University of Glasgow

[2]Instituto Superior Técnico, Technical University of Lisbon

[3]Security and Quantum Information Group, Instituto de Telecomunicações

Our problem

In a nutshell

1. Types-as-processes: a popular approach to discipline the behaviour of concurrent systems
2. Main settings: session types and CCS-like types
3. Session type systems
   require specific language constructs
   provide type safety and progress properties
4. CCS-like type systems
   require specific subtyping relations
   provide safety properties for free, and deadlock freedom

A question

Generic type systems subsumes session type systems?

Session Types (Honda et al.)

The role in programming languages

- Technique for specifying and verifying protocols in concurrent and distributed systems
- Describe the sequence and type of messages that can be sent on a (point-to-point) channel
- Allow certain properties of protocol implementations to be verified by static typechecking

Type systems for different applications

- Idioms of the pi-calculus
- Object-oriented languages
- Service-oriented systems

A session type

Server's view

$$\& \langle \mathsf{service} : ?[\mathsf{int}] \,.\, ![\mathsf{bool}] \,.\, \mathsf{end}, \mathsf{quit} : \mathsf{end} \rangle$$

Behavioural description

- A client can select either service or quit
- From the client's viewpoint, the session has a dual type

Safety property

1. Messages are of the type expected by the receiver
2. Whenever a client selects a service, the server offers a matching service

The generic type system (Igarashi and Kobayashi)

Aim

- A single generic system for the pi-calculus from which numerous specific type systems can be obtained
- Express the common aspects of a range of type systems
- Provides typing rules and type soundness

Technicalities

- Types are abstractions of processes
- Typing rules display correspondence between the structure of processes and the structure of types
- A subtyping relation can be modified in order to obtain specific type systems

A generic type

Server's view

$$\text{service}?[(x)x?[\text{int}]\,.\,x\,!\,[\text{bool}]\,.\,\mathbf{0}]\,.\,\mathbf{0} \;+\; \text{quit}?[]\,.\,\mathbf{0}$$

Behavioural description

- An external choice between service or quit
- (Method) names should be transmitted before to a client

Session processes into generic processes

Challenges

Two features to deal with: polarities and labels

Polarised channels

1. Polarities distinguish between the two endpoints of a channel
2. Communication only occurs between x^+ and x^-

Encoding polarities

1. For each source channel introduce a pair of target channels
2. Put in parallel with the translated process a forwarder between them

Session processes into generic processes

Encoding labels

1. Create a new name for every possible label, send them all and wait in an input-guarded sum
2. Receive a fresh channel for the continuation of the protocol

Encoding input-guarded labelled sums

$$\llbracket x^p \triangleright \{\text{service} : x^p?[y] \,.\, x^p\,![b] \,.\, \mathbf{0}, \text{quit} : \mathbf{0}\}\rrbracket \;=$$
$$(\nu\ \text{service}, \text{quit})\, xp\,![\text{service}, \text{quit}] \,.$$
$$(\text{service}?[x] \,.\, x?[y] \,.\, x\,![b] \,.\, \mathbf{0} \;+\; \text{quit}?[] \,.\, \mathbf{0})$$

Encoding type environments

$$\llbracket x^p : \&\, \langle \text{service} : ?[\text{int}] \,.\, ![\text{bool}] \,.\, \text{end}, \text{quit} : \text{end}\rangle \rrbracket \;=$$
$$xp\,![(\text{service}, \text{quit})($$
$$\text{service}\,![(x)\llbracket x : ?[\text{int}] \,.\, ![\text{bool}] \,.\, \mathbf{0}\rrbracket] \,.\, \mathbf{0} \;\&\; \text{quit}\,![(x)\llbracket x : \mathbf{0}\rrbracket] \,.\, \mathbf{0})]$$

The encoding at work

- Let $Q = x^p \,!\, [y^q] \,.\, P$ and $\Delta = \Gamma, x^p : ![S] \,.\, S', y^q : S$.
- We'll show that $\Delta \vdash Q \;\Rightarrow [\![\Delta]\!] \triangleright [\![Q]\!]$.

1. By (induction) hypothesis, $D = [\![\Gamma, x^p : S']\!] \triangleright [\![P]\!]$.
2. Using the output rule,
   $D1 = xp \,!\, [(z)[\![z : S]\!]] \,.\, (D \mid [yq/z](z)[\![z : S]\!]) \triangleright [\![Q]\!]$
3. By subtyping,
   $D2 = [\![\Gamma]\!] | xp \,!\, [(z)[\![z : S]\!]].([\![x^p : S']\!] | [yq/z](z)[\![z : S]\!]) <: \; D1$
4. Finally, again by subtyping,
   $[\![\Delta]\!] = [\![\Gamma]\!] \mid xp \,!\, [(z)[\![z : S]\!]] \,.\, [\![x^p : S']\!] \mid [\![y^q : S]\!] <: \; D2$

Results

Operational correspondence

- For any well-typed closed session process P, whenever $P \longrightarrow Q$, then $[\![P]\!] \longrightarrow^n [\![Q]\!]$ with $n = 2$ or 4, depending on whether the step is a communication or a selection.
- Reverse direction more complicated to state.

Typing correspondence

- For any session process P, if $\Delta \vdash P$ then $[\![\Delta]\!] \triangleright [\![P]\!]$.
- Let P be a closed session process. If $[\![P]\!]$ is well-typed in the generic type system and no message in P has type $\mathtt{end}$, then P is well-typed as a session process.

Conclusions

1. Session types are a high-level abstraction for structuring inter-process communication
2. Session types applied already for languages other than the π-calculus
   Developing GTSs to other language might not be as easy
3. Proofs of type soundness for session types are straightforward
   Work saved by the generic type soundness theorem is relatively small
4. GTS does not yield typechecking algorithms automatically
   Specific algorithms (for session types) need to be developed anyway