ΗΟΠ and TYPES

NOBUKO YOSHIDA
My Mobile Research Life in UK

1998 May – 1999 Sep
Before starting a post-doc position...

- John Power @ Edinburgh

Matthew is ROYAL

- TOPICS
  - $\mathbb{D} \pi$
  - Subtyping
  - HOπ

Tackle!
Assigning Types to Processes [LICS 2000, I&C]
Matthew Hennessy, NY

Matthew Hennessy, Julian Rathke, NY
Higher-Order π-Calculus

\[ \lambda v \cdot (\lambda x.Q) V \rightarrow Q[V/x] \]

\[ \rho v \cdot \overline{a}[P] \mid a(X). \text{run } X \]

\[ \rightarrow \text{ run } P \rightarrow P \]

where

\[ \text{thunk } P \] = \( \lambda () . P \)

\[ \text{run } = \lambda X . X() \]
Aims of Types / Typechecking

- Using Types to control the effects of Mobile Code / Processes

- Host refuses to execute incoming code unless it conforms to predetermined access policy

\[ a(x:T).P \]

\[ Q \rightarrow \text{Type} \]

STOP
Existing $\lambda/\pi$ Typing System

[1993 - 2000]
Before [YH 00]

$\lambda \rightarrow + \pi \ i o$

$\Sigma ::= \text{unit} \mid \text{nat} \mid \Sigma \rightarrow \rho \mid 6 \mid \text{proc}$

Value Term

$6 ::= (\Sigma^i) \mid (\Sigma^0) \mid (\Sigma^{i0})$

channel Input Output Input-Output

Typing Processes ... Too Simple

$\Gamma \vdash \rho : \text{proc}$

$\Gamma \vdash \rho : \text{proc}$ \quad $\Gamma \vdash \sigma : \text{proc}$

e.g.

$\Gamma \vdash \rho \mid \sigma : \text{proc}$

cf. $\Gamma \vdash M : \Sigma \rightarrow \rho$
Problem

$c(X: \text{proc}). \text{run } X$

where $\text{proc} = \text{unit} \rightarrow \text{proc}$

Any Process is welcome

We are very different

P $a(X). \overline{b}(X)$
Q $b(X). \overline{a}(X)$
R $d(\text{string})$
the Idea is simple but ...

quite a bit of work


Channels appear both in Types and Processes

\[ F = \lambda x. \lambda (X : \{x : (z)^0, b : (z)^0\}) \cdot (\text{run } X \mid \overline{a} \langle 1 \rangle \mid b \langle 2 \rangle) \]

\[ F a \rightarrow \lambda (X : \{a : (z)^0, b : (z)^0\}) \cdot (\text{run } X \mid \overline{b} \langle 1 \rangle \mid b \langle 2 \rangle) \]

\[ F b \rightarrow \lambda (X : \{b : (z)^0\}) \cdot (\text{run } X \mid \overline{b} \langle 1 \rangle \mid b \langle 2 \rangle) \]

\[ \Rightarrow \text{ Kinding / Dependency Types} \]

\[ \text{[ Yoshida - Hennessy 2000]} \]

- Functional Channel Dependency
- Channel Dependency (Non-determinism)
- Existential Types (Scope-Opening)

Another bit of work
Higher Order π-calculus

Syntax

\[ P, Q ::= V \quad \text{value} \]
\[ 0 \quad \text{nil} \]
\[ P \mid Q \quad \text{parallel} \]
\[ \bar{u} < v_1, \ldots, v_n > \quad \text{OUTPUT} \]
\[ u(x; z_1, \ldots, x; z_m) P \quad \text{INPUT} \]
\[ !P \quad \text{Replication} \]
\[ (v_a; v_0) P \quad \text{Restriction} \]
\[ P Q \quad \text{Application} \]

\[ V, W ::= \lambda (x; z) P \quad \lambda\text{-abst} \]
\[ 1, 2, \ldots, (), \ldots \quad \text{constant} \]
\[ x, y, z, \ldots \quad \text{variables} \]
\[ a, b, c, \ldots \quad \text{channels/names} \]
Types

Term
$\mathbb{Z} ::= \text{unit, nat}$
\[ \pi : \mathbb{Z} \rightarrow \mathbb{Z} \]
\[ \pi (x : \mathbb{Z}) \mathbb{Z} \]
\[ \{ \Delta \} \]
\[ 6 \]

Channel
$6 ::= (\pi [\bar{x} : \mathbb{Z} \leftarrow])^p$
\[ \exists [\bar{z} : \mathbb{Z} \leftarrow] (p) \]
\[ \langle 61, 60 \rangle \]
\[ p ::= I \mid 0 \]
Typing System

(Zero) \[
\Gamma \vdash \text{Env} \quad \Rightarrow \quad \Gamma \vdash \text{0} \odot [\text{}] \\
\]

(Par) \[
\Gamma \vdash P @ [\Delta] \quad \Gamma \vdash Q @ [\Delta'] \\
\Rightarrow \quad \Gamma \vdash P ! Q @ [\Delta \cup \Delta'] \\
\]

(Res) \[
\Gamma, a : 6 \vdash P @ [\Delta, a : 6] \\
\Rightarrow \quad \Gamma \vdash (v a : 6) P @ [\Delta] \\
\]

(Rep) \[
\Gamma \vdash P @ [\Delta] \\
\Rightarrow \quad \Gamma \vdash ! P @ [\Delta] \\
\]
Examples

\[ \text{FW}(ab) \vdash a \ (x:Z), \overline{b} \ <x>: [a:(Z)^1, b:(Z)^0] \]

\[ \text{FW}(ba) \vdash b \ (x:Z), \overline{a} \ <x>: [b:(Z)^1, a:(Z)^0] \]

\[ \text{par} \ \vdash \text{FW}(ab) \mid \text{FW}(ba) : [a:(Z)^{10}, b:(Z)^{10}] \]

because

\[ [a:(Z)^1, b:(Z)^0] \sqcup [a:(Z)^0, b:(Z)^1] \]
\[ = [a:(Z)^{10}, b:(Z)^{10}] \]

restriction \[ a:(Z)^{10}, b:(Z)^{10} \vdash \text{FW}(ab) : [a:(Z)^1, b:(Z)^0] \]

\[ \downarrow \]

\[ a:(Z)^{10} \vdash (\nu b) \text{FW}(ab) : [a:(Z)^1] \]

\[ \downarrow \]

\[ \vdash (\nu a b) \text{FW}(ab) : [] \]
Script Server

\[ FW = \lambda (x:z). \lambda (y:z). \bar{x} (z). \bar{y} (z) \]
forwarder

\[ \overline{c} \langle FW \rangle \rightarrow \overline{c} \langle FW \rangle \]
Server

\[ c(y). yab \]
client 1

\[ FW(ab) \]
\[ : [a:(z)^i, b:(z)^0] \]
C1 evolves into

\[ \overline{c} \langle FW \rangle \rightarrow \overline{c} \langle FW \rangle \]
Server

\[ c(y). yba \]
client 2

\[ FW(ba) \]
\[ : [a:(z)^0, b:(z)^i] \]
C2 evolves into

(FZ)^i \rightarrow (FZ)^0 \rightarrow proc

Func Dep [YHOO]
\[ \Pi (x:(FZ)^i) \Pi (y:(FZ)^0) [x: (FZ)^i, y: (FZ)^0] \]
Existential Types for Scope Opening

Client (A) wishes to execute code $P$ and to get ack $\bar{c}(V)$ at the time $P$ is executed at the remote location (B)

\[
\nu \pi_c \quad b(P | \bar{c}(V)) \mid c(x). R
\]

(1) $c$ is private
(2) $V$ must not be touched (i.e. compromised)
Existential Types for Scope Opening

\[(B)\]
\[b(x).\text{run} X\]

\[(VC)\]
\[\overline{\exists \langle v \rangle} \mid \text{private name}\]
\[c(y).R\]

\[(VC)\]
\[p \mid \overline{\exists \langle v \rangle} \mid c(y).R\]

\[(VC)\]
\[p', \downarrow\]

\[\text{previous system}\]

\[\text{channel existential types}\]

\[\text{[POPL 04]}\]

\[\left( \exists [x:6]\right) \Gamma, x:6^+\]
\[\text{anonymous channel of type 6}\]
Typing System for $\exists$

\[(\text{In}^\exists)\]
\[
\Gamma \vdash a : (\exists [x:6] z)^1
\]
\[
\Gamma, \{x:6, x:7\} \vdash P \circ \Delta, x:6
\]
\[
\Gamma \vdash a(x: \exists [x:6] z). P \circ \Delta, a : (\exists [x:6] z)^1
\]

\[(\text{Out}^\exists)\]
\[
\Gamma \vdash a : (\exists [x:6] z)^0
\]
\[
\Gamma \vdash \{ c, V \} : \exists [x:6] z
\]
\[
\Gamma \vdash \overline{a} < V \circ \Delta, a : (\exists [x:6] z)^0, c:6
\]

Proposition (Minimum Interface)

\[
\Gamma \vdash P \circ [\Delta_i] \Rightarrow \exists ! \Delta_{\text{min}} \leq \Delta_i
\]
\[
\text{s.t. } \Gamma \vdash P \circ [\Delta_{\text{min}}]
\]
Main Theorems

Subject Reduction
\[ \Gamma \vdash P : \Delta, \ P \rightarrow P' \Rightarrow \Gamma \vdash P' : \Delta \]

Type Safety
\[ \Gamma \vdash P : [\Delta] \Rightarrow P \xrightarrow{\Gamma', [\Delta]}_{err} \]

where
\[ P \xrightarrow{\Gamma', [\Delta]}_{err} \]
means
P can use at most resources in \( \Delta \)

Consequence:
\[ a(x : [\Delta]). P \mid \overrightarrow{a} \langle \Gamma' \rangle \xrightarrow{\Gamma', \pi}_{err} \]
if \( \Gamma \nvdash R : [\Delta] \)

\[ \Delta' \]
Encapsulation of Higher-Order Code by Hidden Name

Theorem

\[ \Gamma \vdash P : \Delta \] and \( \text{fv}(P) = \emptyset \)
and \[ \Gamma \vdash a : \exists (x : e) \uparrow x : e^{-1} \uparrow \]

Then

\[ P \xrightarrow{a \langle \exists (v) \rangle} P' \Rightarrow P' \xrightarrow{\_} \]

Mobile Code bound by \( \exists \)-name is eventually returned to the sender without being touched by the receiver
Ocean Observatory Initiative

a use of multiparty session types as a runtime monitor
ESOP 2009: Asynchronous Subtyping for Session Types

Mostrous, Honda, NY
PPDP'14: Preciseness of Session Subtyping
[Tzu-Chun Chen, Mariangiola Dezani and NY]
Session Type checking

- Static analyser
- Does source code conform to specification?
- Extract session type from code
  - Based on usage of API
  - Based on program flow control
- Compare w/ endpoint protocol
Session Type checking: Asynchronous optimisation

- Protocols designed safe
- Naive impl. inefficient
- Asynchronous impl.
  - Non-blocking send
  - Blocking receive
- Overlap send/recv operations
- Safety by async. subtyping
  [Mostrous et al., ESOP’09]
**Example: Ring topology in Pabble**

```plaintext
1  global protocol Ring(role Worker[1..N]) {
2    rec LOOP {
3      Data(int) from Worker[i:1..N-1] to Worker[i+1];
4      Data(int) from Worker[N] to Worker[1];
5      continue LOOP;
6    }
7  }
```
Typed Semantics in $\Pi$ 1991 →

IO-subtyping, Linear types, Secure Information Flow, ...

Many Hennessy’s papers study typed/environment bisimulation
GLOBALLY GOVERNED
SESSION SEMANTICS

Dimitrios Kouzapas
Glasgow

Nobuko Yoshida
Imperial College London

CONCUR’13
Multiparty Compatibility in Communicating Automata

Synthesis and Characterisation of Multiparty Session Types
1. Deterministic
2. No-Mixed State
3. Compatible

[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.
http://www.zdlc.co/faq/

WHAT DOES ZDLC DO?

Professor Steve Ross-Talbot
Managing Director, ZDLC BU
Cognizant Technical Services
Zero Deviation Life Cycle Platform

Adapters
- C/C++
- JAVA
- MAINFRAME
- .NET
- IBM BPM
- COBOL
- TANDEM
- ORACLE DB
- MQs

Reverse Engineering Core Module
- Configuration Matrix
- Crawler
- Generic Parser
- Analyser

Report Composer
- Reporting Module

System Behaviour
- JVM Logs, Application Logs
- System logs, DB Logs

SCRIBBLE

UML & BPMN2 Model
POPL 2015: From Communicating Machines to Graphical Choreographies [Lange, Tuosto, NY]
HAPPY BIRTHDAY

Matthew