A linear account of session types in the pi calculus

joint work with Vasco T. Vasconcelos

Marco Giunti

University IUAV of Venice

CONCUR, September 2 2010, Paris

Session Types

- Describe a protocol between a service provider and a client

- Introduced for the pi calculus and now embedded also in other paradigms based on message passing
  - functional programming
  - object oriented programming

- Idea: allowing typing of channels by using structured sequences of types as output,output,input,..

$$!\text{Integer} \,.\, !\,\text{Boolean} \,.\, ?\,\text{Boolean} \,.\, \text{end}$$

Session types in the pi calculus

- In [HVK Esop'98] a typing discipline for structured programming is introduced for a dialect of pi calculus

- Session channels are used to abstract binary sessions and are distinguished from standard pi calculus channels or names

- Session initiation arises on names

- Fidelity of sessions is guaranteed by a typing system enforcing a session channel to be used at most by two threads with opposite capabilities (e.g. input/output)

Discussion

- In the original system and recent works session delegation is restricted to bound output

$$\overline{x}\, k.P \mid x(k).Q \rightarrow P \mid Q$$

- Communication mechanism of the pi calculus breaks subject reduction

- Decoration of channel end-points is the de-facto workaround [GH Acta'05]

$$\overline{x^+}\, y^p.P \mid x^-(z).Q \rightarrow P \mid Q[y^p/z]$$

- Distinction between names and session channels of [HVK98] leads to duplicate typing rules

This talk

- Do not distinguish session channels from names
- Do not use polarities or double binders
- That is: we use standard pi calculus
- Annotate session types with qualifiers
  - lin for linear use
  - un for unrestricted or shared use
- Introduce a type construct that describes the two ends of a same channel

Expressiveness

- We encode
  1. linear lambda calculus [Walker&05]
  2. linear pi calculus [KPT TOPLAS'99]
  3. pi calculus with polarities [GH Acta'05]
- We prove operational and typing correspondence

Example: Online Petition

- Creators of the petition receive from service a session channel to
  1. provide the petition title
  2. provide the petition description
  3. sign zero or more times the petition

- The session may be *delegated* to let subscribers sign the petition

$$\textit{petition}(p).\overline{p}\,\mathsf{title}.\overline{p}\,\mathsf{description}.\overline{p}\,\mathsf{signature}.(\overline{c_1}\,p \mid \ldots \mid \overline{c_n}\,p)$$

- Limitation: not clear how to type this process with current systems

- Concurrent distribution of the same channel problematic

Typing the petition client

- Petition channel used first in linear mode then in unrestricted mode
- Steps:
  1. Provide the petition title (linear mode)
  2. Provide the petition description (linear mode)
  3. Sign the petition zero or more times (unrestricted mode)

$$\textit{petition}(p).\overline{p}\,\mathsf{title}.\overline{p}\,\mathsf{description}.\overline{p}\,\mathsf{signature}.(\overline{c_1}\,p \mid \ldots \mid \overline{c_n}\,p)$$

- End point session type for channel p is

$$\mathsf{lin}\,!\textit{Title}.\mathsf{lin}\,!\textit{Description}.S \quad \text{where} \quad S = \mathsf{un}\,!\textit{Signature}.S$$

- Unrestricted recursive type S allows for concurrent distribution of petition channel

Pi calculus

b ::=		Booleans:	P ::=		Processes:
	true	true		$\overline{x}\, v.P$	output
	false	false		$x(x).P$	input
v ::=		Values:		$P \mid P$	composition
	b	boolean value		if v then P else P	conditional
	x	variable		$(\nu x)P$	restriction
				$!P$	replication
				$\mathbf{0}$	inaction

$$\overline{x}\, v.P \mid x(y).Q \;\rightarrow\; P \mid Q[v/y] \qquad \frac{P \;\rightarrow\; Q}{P \mid R \;\rightarrow\; Q \mid R}$$

$$\text{if true then } P \text{ else } Q \;\rightarrow\; P \qquad \text{if false then } P \text{ else } Q \;\rightarrow\; Q$$

$$\frac{P \;\rightarrow\; Q}{(\nu x)P \;\rightarrow\; (\nu x)Q} \qquad \frac{P \equiv P' \quad P' \;\rightarrow\; Q' \quad Q' \equiv Q}{P \;\rightarrow\; Q}$$

Types

- A type T is
  - bool for boolean values
  - S for *end point type* describing one channel end
  - (S, S) for *channel type* describing both channel ends
- End point types S are
  - $q\,p$ for qualified session types
  - $\mu a.S$ and a for recursive end point types

Qualified Session Types

- We qualify session types p with $q = \text{lin} \mid \text{un}$
  - lin p: channel is used exactly once at given type
  - un p: channel is used zero or more times
- Session types p are
  - $?T.S$: channel waits for a value of type T then continues as end point type S
  - $!T.S$: channel sends a value of type T then continues as end point type S
  - end: no further interactions are possible

Type Duality

- Duality plays central role
- Qualifiers preserved

$$\overline{q\,?T.S} = q\,!T.\overline{S} \qquad \overline{q\,!T.S} = q\,?T.\overline{S}$$

$$\overline{q\ \mathsf{end}} = q\ \mathsf{end} \qquad \overline{\mu a.S} = \mu a.\overline{S}$$

$$\overline{a} = a$$

A type for the petition service

- Service accepting unbounded signatures

$$S = (\nu p)\,\overline{\textit{petition}}\,p.p(y).p(z).!p(\textit{signature})$$

- The channel used for the petition is described by a channel type

$$p : (\mathsf{lin}\,!\textit{Title}.\mathsf{lin}\,!\textit{Description}.S,\ \mathsf{lin}\,?\textit{Title}.\mathsf{lin}\,?\textit{Description}.\overline{S})$$

$$S = \mu a.\mathsf{un}\,!\textit{Signature}.a \quad (*!\textit{Signature}\text{ for short})$$

- Notice that the channel type is composed by dual end point types

$$\overline{\mathsf{lin}\,!\textit{Title}.\mathsf{lin}\,!\textit{Description}.S} = \mathsf{lin}\,?\textit{Title}.\mathsf{lin}\,?\textit{Description}.\overline{S}$$

Context splitting

- We define a splitting operation over type environments
  - A context or type environment Γ is $\emptyset$ or $\Gamma, x : T$

- Linear types are placed in one context, unrestricted types in both [Walker&05]

$$\emptyset = \emptyset \cdot \emptyset \qquad \frac{\Gamma = \Gamma_1 \cdot \Gamma_2 \qquad T = \text{un}\, p \text{ or } (\text{un}\, p_1, \text{un}\, p_2)}{\Gamma, x\colon T = (\Gamma_1, x\colon T) \cdot (\Gamma_2, x\colon T)}$$

$$\frac{\Gamma = \Gamma_1 \cdot \Gamma_2 \qquad T = \text{lin}\, p \text{ or } (\text{lin}\, p_1, \text{lin}\, p_2)}{\Gamma, x\colon T = (\Gamma_1, x\colon T) \cdot \Gamma_2}$$

$$\frac{\Gamma = \Gamma_1 \cdot \Gamma_2 \qquad T = \text{lin}\, p \text{ or } (\text{lin}\, p_1, \text{lin}\, p_2)}{\Gamma, x\colon T = \Gamma_1 \cdot (\Gamma_2, x\colon T)}$$

Splitting channel types

- Linear channel types are split in two linear end points

$$\frac{\Gamma = \Gamma_1 \cdot \Gamma_2}{\Gamma, x\colon (\mathsf{lin}\, p_1, \mathsf{lin}\, p_2) = (\Gamma_1, x\colon \mathsf{lin}\, p_1) \cdot (\Gamma_2, x\colon \mathsf{lin}\, p_2)}$$

- Mixed channel types are split into a mixed channel type and an unrestricted end point

$$\frac{\Gamma = \Gamma_1 \cdot \Gamma_2}{\Gamma, x\colon (\mathsf{lin}\, p_1, \mathsf{un}\, p_2) = (\Gamma_1, x\colon (\mathsf{lin}\, p_1, \mathsf{un}\, p_2)) \cdot (\Gamma_2, x\colon \mathsf{un}\, p_2)}$$

$$\frac{\Gamma = \Gamma_1 \cdot \Gamma_2}{\Gamma, x\colon (\mathsf{lin}\, p_1, \mathsf{un}\, p_2) = (\Gamma_1, x\colon \mathsf{un}\, p_2) \cdot (\Gamma_2, x\colon (\mathsf{lin}\, p_1, \mathsf{un}\, p_2))}$$

Splitting on parallel composition

- Rule for composition splits environment in two threads

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 \cdot \Gamma_2 \vdash P_1 \mid P_2}$$

- Typing the petition protocol

$$\frac{\Gamma_1, p : \mathsf{lin}\,!\mathit{Title}.\mathsf{lin}\,!\mathit{Descr}.S \vdash P_1 \qquad \Gamma_2, p : \mathsf{lin}\,?\mathit{Title}.\mathsf{lin}\,?\mathit{Descr}.\overline{S} \vdash P_2}{\Gamma, p : (\mathsf{lin}\,!\mathit{Title}.\mathsf{lin}\,!\mathit{Descr}.S, \mathsf{lin}\,?\mathit{Title}.\mathsf{lin}\,?\mathit{Descr}.\overline{S}) \vdash P_1 \mid P_2}$$

- $P_1 = \overline{p}\,\text{title}.\overline{p}\,\text{description}.\overline{p}\,\text{signature}.(\overline{c_1}\,p \mid \cdots \mid \overline{c_n}\,p)$

- $P_2 = p(y).p(z).!p(w)$

Splitting on sending values

- Rule for sending values splits the sent capability

$$\frac{\Gamma_1 \vdash v\colon T \qquad \Gamma_2, x\colon S \vdash P \qquad q = \mathsf{un} \Rightarrow q!T.S = S}{\Gamma_1 \cdot (\Gamma_2, x\colon q\,!T.S) \vdash \overline{x}\, v.P}$$

- Typing the petition service

$$\frac{\textit{petition} : *!T, p : \overline{T} \vdash p(y).p(z).!p(w)}{\textit{petition} : *!T, p : (T, \overline{T}) \vdash \overline{\textit{petition}}\, p.p(y).p(z).!p(w)}$$

- $T = \mathsf{lin}\,!\textit{Title}.\mathsf{lin}\,!\textit{Descr}.*!\textit{Signature}$

- $\overline{T} = \mathsf{lin}\,?\textit{Title}.\mathsf{lin}\,?\textit{Description}.*?\textit{Signature}$

Rules for channel types

- Rule for sending with channel type

$$\frac{\Gamma_1 \vdash v\colon T \qquad \Gamma_2, x\colon (S, S') \vdash P \qquad q = \textsf{un} \Rightarrow q!T.S = S}{\Gamma_1 \cdot (\Gamma_2, x\colon (q\,!T.S, S')) \vdash \overline{x}\, v.P}$$

- E.g., counter-example to subject reduction for pi calculus with free communication [DCDMY09]

$$\overline{x}\, v \mid x(y).\overline{v}\,\textsf{true}.y(z) \longrightarrow \overline{v}\,\textsf{true}.v(z)$$

- Typing the redex

$$\frac{v : (\textsf{un end}, \textsf{lin}\,?\textsf{bool}.\textsf{un end}) \vdash v(z)}{v : (\textsf{lin}\,!\textsf{bool}.\textsf{un end}, \textsf{lin}\,?\textsf{bool}.\textsf{un end}) \vdash \overline{v}\,\textsf{true}.v(z)}$$

Rules for input processes

- End point

$$\frac{\Gamma, x\colon S, y\colon T \vdash P \qquad (*)}{\Gamma, x\colon q?T.S \vdash x(y).P}$$

- Channel

$$\frac{\Gamma, x\colon (S, S'), y\colon T \vdash P \qquad (*)}{\Gamma, x\colon (q?T.S, S') \vdash x(y).P}$$

$$(*)\ q = \mathsf{un} \Rightarrow q?T.S = S$$

- Typing the petition client

$$\frac{\Gamma, \textit{petition}\colon *?T, p\colon T \vdash \overline{p}\,\mathsf{t}.\overline{p}\,\mathsf{d}.\overline{p}\,\mathsf{s}.(\overline{c_1}\,p \mid \cdots \mid \overline{c_n}\,p)}{\Gamma, \textit{petition}\colon *?T \vdash \textit{petition}(p).\overline{p}\,\mathsf{t}.\overline{p}\,\mathsf{d}.\overline{p}\,\mathsf{s}.(\overline{c_1}\,p \mid \cdots \mid \overline{c_n}\,p)}$$

$T = \mathsf{lin}\,!\textit{Title}.\mathsf{lin}\,!\textit{Descr}.*!\textit{Signature}$

Consuming resources

- Linear resources are eventually consumed

$$\frac{\mathsf{un}(\Gamma)}{\Gamma \vdash \mathbf{0}}$$

- Distributing the petition channel

$$\frac{c_1 : \mathsf{un\ end}, p : S \vdash \mathbf{0} \quad \cdots \quad c_n : \mathsf{un\ end}, p : S \vdash \mathbf{0}}{c_1 : \mathsf{lin}!S.\mathsf{un\ end}, \cdots, c_n : \mathsf{lin}!S.\mathsf{un\ end}, p : S \vdash \overline{c_1}\, p.\mathbf{0} \mid \cdots \mid \overline{c_n}\, p.\mathbf{0}}$$

- $S = *!$*Signature*

- Replication for unrestricted resources

$$\frac{\Gamma \vdash P \qquad \mathsf{un}(\Gamma)}{\Gamma \vdash !P}$$

Balanced channel types

- New channel generated at balanced channel type

$$\frac{\Gamma, x\colon (S, \overline{S}) \vdash P}{\Gamma \vdash (\nu x)P}$$

- Balancing preserve typability under reduction

- Counter-example:

$$x\colon (\mathsf{lin!end.end}, \mathsf{lin?bool.end}) \vdash x(z).\mathsf{if}\ z\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ \mathbf{0} \mid (\nu y)\overline{x}\,y$$

$$x(z).\mathsf{if}\ z\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ \mathbf{0} \mid (\nu y)\overline{x}\,y \longrightarrow (\nu y)\mathsf{if}\ y\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ \mathbf{0}$$

- The purpose of balancing is to ensure that the type of y is that of z

Subject reduction

- Let $\Gamma_1 \vdash P_1$ with Γ_1 balanced. If $P_1 \rightarrow P_2$ then $\Gamma_2 \vdash P_2$ with Γ_2 balanced

- Relies on a stronger result

  1. $\Gamma_2 = \Gamma_1$ or
  2. $\Gamma_1 = \Gamma', x\colon (\mathsf{lin}?T.S, \mathsf{lin}!T.\overline{S})$ and $\Gamma_2 = \Gamma', x\colon (S, \overline{S})$

Embedding pi with polarities[GH Acta'05]

- Standard channel types mapped into shared channel types

$$[\![\hat{\ } T\,]\!] = (*?[\![T]\!], *![\![T]\!])$$

- Session types mapped into linear types

$$[\![?T.S]\!] = \mathsf{lin}?[\![T]\!].[\![S]\!] \qquad [\![!T.S]\!] = \mathsf{lin}![\![T]\!].[\![S]\!]$$

- Context rules in given order

$$[\![\Gamma, x^+ \colon S, x^- \colon S']\!] = [\![\Gamma]\!], x \colon ([\![S]\!], [\![S']\!])$$
$$[\![\Gamma, x^p \colon T]\!] = [\![\Gamma]\!], x \colon [\![T]\!]$$

- Polarity-Pi to Pi Correspondence
  - If $\Gamma \vdash_{\mathsf{p}} P$ then $[\![\Gamma]\!] \vdash \mathrm{erase}(P)$.
  - If $P \rightarrow_{\mathsf{p}} Q$, then $\mathrm{erase}(P) \rightarrow \mathrm{erase}(Q)$.

Encoding linear pi calculus [KPT'99]

- Linear types are used once then discarded

$$[\![\text{lin i}T]\!] = \text{lin}?[\![T]\!].\text{un end} \qquad [\![\text{lin o}T]\!] = \text{lin}![\![T]\!].\text{un end}$$

- Unrestricted types maintain their behaviour

$$[\![\text{un i}T]\!] = *?[\![T]\!] \qquad [\![\text{un o}T]\!] = *![\![T]\!]$$

- i/o types mapped into channel types!

$$[\![q\ \text{io}\ T]\!] = ([\![q\ \text{i}T]\!], [\![q\ \text{o}T]\!])$$

- Linear-Pi to Pi Correspondence
  - If $\Gamma \vdash_{\text{l}} P$ then $[\![\Gamma]\!] \vdash P$.

Encoding linear lambda calculus

- Call-by-value linear lambda calculus [Walker&05] into polyadic pi
- Types: $T \; = \; q\,\mathsf{bool} \mid qT \to T \qquad q = \mathsf{lin} \mid \mathsf{un}$
- Unrestricted functions mapped into unrestricted channel type

$$[\![\mathsf{un}\, T_1 \to T_2]\!] = (*?X, *!X)$$

  - input for the function proper, output for clients
  - X is pair: (1) function's argument (2) linear channel for result

$$X = \langle [\![T_1]\!], \mathsf{lin}![\![T_2]\!].\mathsf{un\ end}\rangle$$

- Linear functions into linear channel type carrying pair X

$$[\![\mathsf{lin}\, T_1 \to T_2]\!] = (\mathsf{lin}?X.\mathsf{un\ end}, \mathsf{lin}!X.\mathsf{un\ end})$$

Correspondence

- Encoding of terms small variation of Milner's encoding [JFP'05]

- Linear-lambda to pi correspondence

1. If $\Gamma \vdash_\lambda M : T$, then $[\![\Gamma]\!], p\colon \mathsf{lin}![\![T]\!].\mathsf{un\ end} \vdash [\![M]\!]_p$.

2. If $(S; M) \rightarrow_\lambda (S'; M')$, then $[\![S; M]\!]_p \rightarrow^* [\![S'; M']\!]_p$.

Conclusions

- We introduced a type construct that describes the two ends of the same channel

- Linear channels could evolve to shared channels

- Future work

  - Algoritmic type-checking almost completed

  - Avoid deadlocks