A linear type system for pi calculus

joint work with Vasco T. Vasconcelos

Marco Giunti

INRIA & LIX, École Polytechnique Palaiseau

Behavorial Types, April 19 2010, Lisboa

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

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

- Communication mechanism of the pi calculus breaks subject reduction

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

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

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

What we have done

- Remove distinction among session channels and 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 use
- Introduce a type construct that describes the two ends of a same channel

Types

- Types T
  - S for *end point type* describing one channel end
  - (S, S) for *channel type* describing both channel ends
- End point types S are
  - lin p linear channel used exactly once
  - un p channel is used zero or more times
  - $\mu a.S$ and a for recursive end point types
- Session types p are
  - $?T.S$: waits for value of type T then continues as S
  - $!T.S$: sends a value of type T then continues as S
  - end: no further interactions are possible

Example: event scheduling

1. Create poll
   - provide the title for the meeting
   - provide a provisional date
2. Invite participants

- Pi calculus: send request to create poll / receive poll channel

$$\overline{\textit{poll}}\langle y\rangle.y(p).(\overline{p}\langle \text{Workshop}\rangle.\overline{p}\langle \text{19April}\rangle.(\overline{z_1}\langle p\rangle \mid \cdots \mid \overline{z_n}\langle p\rangle))$$

- Challenge: concurrent distribution of the poll channel

Session type for the poll

- Poll channel used first in linear mode then in unrestricted mode

- Steps:

  1. Send a title for the poll (linear mode)

  2. Send a date for the poll (linear mode)

  3. Distribute the poll (unrestricted mode)

$$y(p).(\overline{p}\langle \text{Workshop} \rangle.\overline{p}\langle \text{19April} \rangle.(\overline{z_1}\langle p \rangle \mid \cdots \mid \overline{z_n}\langle p \rangle))$$

- End point session type for channel p is

$$\text{lin}\,!\text{string}.\text{lin}\,!\text{date}.*S \quad \text{where} \quad *S = \text{un}\,!\text{date}.*S$$

- Recursive unrestricted type S allows distribution of poll channel

Type for the scheduling service

- Service: instantiation generates poll

$$\textit{Service} = !\textit{poll}(w).(\nu p : (S_1, S_2))\,(\overline{w}\langle p\rangle.p(t).p(d).!p(d))$$

$$S_1 = \mathsf{lin}\,?\mathsf{string}.\mathsf{lin}\,?\mathsf{date}.\,{*}\mathsf{un}\,?\mathsf{date}$$

$$S_2 = \mathsf{lin}\,!\mathsf{string}.\mathsf{lin}\,!\mathsf{date}.\,{*}\mathsf{un}\,!\mathsf{date}$$

- Poll channel is **split**:

1. One channel end sent to the invoker

2. The other channel end used in the continuation

Context splitting

- Type system $\Gamma \vdash P$ based on context splitting $\Gamma_1 \cdot \Gamma_2$
- Unrestricted types are copied into both contexts
- Linear types are placed in one of the two resulting contexts

$$\frac{\Gamma_1, p : S_2 \vdash p : S_2 \quad \Gamma_2, w : \mathsf{end}, p : S_1 \vdash p(t).p(d).!p(d) \quad \Gamma = \Gamma_1 \cdot \Gamma_2}{\Gamma, w : \mathsf{lin}\,!S_2.\mathsf{end}, p : (S_1, S_2) \vdash \overline{w}\langle p \rangle.p(t).p(d).!p(d)}$$

Subject reduction

- Γ balanced, $\Gamma \vdash P$, $P \;\rightarrow\; P'$ imply $\Gamma' \vdash P'$ with Γ' balanced

- Interesting case: $(q\,?T.S_1, q\,?T.S_2)$ is balanced if both T and (S_1, S_2) are balanced

- Purpose of balancing is to preserve soundness of exchange

$$\Gamma = x : (\mathsf{lin}\,?(*!\mathsf{bool}).\mathsf{un\ end}, \mathsf{lin}\,!(\mathsf{un\ end}).\mathsf{un\ end}), y : \mathsf{un\ end}$$

$$\Gamma \vdash x(z).\overline{z}\langle\mathsf{true}\rangle \mid \overline{x}\langle y\rangle$$

$$x(z).\overline{z}\langle\mathsf{true}\rangle \mid \overline{x}\langle y\rangle \;\rightarrow\; \overline{y}\langle\mathsf{true}\rangle$$

$$x : (\mathsf{un\ end}, \mathsf{un\ end}), y : \mathsf{un\ end} \not\vdash \overline{y}\langle\mathsf{true}\rangle$$

SR at work

- Receiving of a session already known

$$\overline{x}\langle v\rangle \mid x(y).\overline{v}\langle \mathsf{true}\rangle.y(z) \rightarrow \overline{v}\langle \mathsf{true}\rangle.v(z)$$

- Typing the redex

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

Algorithm

- Type system $\vdash$ cannot be implemented directly
- Main difficulty is split operation
- We avoid split by
  1. passing entire context for the judgement
  2. mark linear types consumed in the derivation as *unusable*

Type checking

- Algorithm relies on several patterns of checking function

$$\mathsf{fun\ check}(g : \mathsf{context}, \mathsf{p} : \mathsf{process}) : \mathsf{context}$$

- Context in input is balanced

  1. patterns are non ambiguous

  2. no backtracking is needed

- Context in output has **void** marks in place of consumed types

- Top-level call accepts process if check returns unrestricted context

$$\mathsf{fun\ typeCheck}(g : \mathsf{context}, \mathsf{p} : \mathsf{process}) : \mathsf{bool}$$

Checking the service

- Poll delegation: type for delegation channel $T = \text{lin}\,!S_2.\text{un end}$

$$\text{check}(\Gamma, w : T, p : (S_1, S_2)\ , \overline{w}\langle p\rangle.P) =$$
$$\text{let val } d = \text{check}(\Gamma, w : \text{un end}, p : (S_1, \circ), P)$$
$$\text{in if } d = d', w : M \text{ and } M = \circ, \text{un}\, p \text{ then } d', w : \circ$$

- Call for the continuation by setting delegated end point for the poll to **void** (noted $\circ$)

- Linear use of channel must be consumed within the continuation (condition $M = \circ, \text{un}\, p$)

- Returned context obtained by setting to void the unrestricted type for the channel

Checking the continuation

- Linear receiving of the date: $S_1 = \mathsf{lin}\,?\mathsf{string}.\mathsf{lin}\,?\mathsf{date}.*\mathsf{un}\,?\mathsf{date}$

$$\mathsf{check}(\Gamma, p : (S_1, N)\,,\; p(t).P) =$$
$$\quad \mathsf{let\ val}\ d = \mathsf{check}(\Gamma, p : \mathsf{lin}\,?\mathsf{date}.*\mathsf{un}\,?\mathsf{date}, t : \mathsf{string}, P)$$
$$\quad \mathsf{in\ if}\ d = d', p : M\ \mathsf{and}\ M = \circ, \mathsf{un}\,p\ \mathsf{then}\ d', p : (\circ, N)$$

- Checking of the continuation invoked by passing *one* channel end

- Linear use of channel must be consumed within the continuation (condition $M = \mathsf{un}\,p, \circ$)

- Returned context re-builds channel type by setting used channel end to void

Checking the scheduling protocol

- Protocol described by concurrent execution of

$$\textit{Service} = !\textit{poll}(w).(\nu p)\,(\overline{w}\langle p\rangle.p(t).p(d).!p(d))$$

$$\textit{Invoker} = \overline{\textit{poll}}\langle y\rangle.y(p).(\overline{p}\langle \text{Workshop}\rangle.\overline{p}\langle \text{19April}\rangle.(\overline{z_1}\langle p\rangle \mid .. \mid \overline{z_n}\langle p\rangle))$$

- Type checking

$$\text{check}(\Gamma\,,\ \textit{Service} \mid \textit{Invoker}) =$$
$$\text{check}(\ \textit{Invoker}\,,\ \text{check}(\Gamma\,,\ \textit{Service})\ )$$

- Preservation of structural congruence

$$\text{check}(\Gamma\,,\ \textit{Invoker} \mid \textit{Service}) = \text{check}(\Gamma\,,\ \textit{Service} \mid \textit{Invoker})$$

Algoritmic soundness

- The algorithm is sound
  - typeCheck(Γ, P) implies $\Gamma \vdash P$

- Completeness missing since $\vdash$ permits to infer
  - $\Gamma, x : (\mathsf{lin}\,?T.S_1, \mathsf{lin}\,!T.S_2) \vdash \overline{x}\langle v\rangle.C[x(y).P]$
  - $\Gamma, x : (\mathsf{lin}\,?T.S_1, \mathsf{lin}\,!T.S_2) \vdash x(y).C[\overline{x}\langle v\rangle.Q]$
  - $\Gamma, x : (\mathsf{lin}\,?T.S_1, \mathsf{lin}\,!T.S_2) \vdash \overline{x}\langle x\rangle.P$

- Claim: processes in these judgements are deadlocked

Towards algoritmic completeness

- Proof transformation: $\Gamma_1 \vdash P_1$ transformed in $\Gamma_2 \vdash P_2$
- Construction: $\Gamma, x : (\text{lin}\ ?T.S_1, \text{lin}\ !T.S_2) \vdash \overline{x}\langle v \rangle.Q$ substituted in the derivation tree for $\Gamma_1 \vdash P_1$ with $\emptyset \vdash \mathbf{0}$
- Typed equivalence: $\Gamma_1 \triangleright P_1$ and $\Gamma_2 \triangleright P_2$ have same *behavior*
  - $\Gamma \triangleright P$ is typed configuration such that $\Delta \vdash P$ and $\Gamma \cdot \Delta$ defined
  - Γ is less informative typed observer allowing moves of P
- Semantic completeness: typeCheck (Γ_2, P_2)

Conclusions

- We introduced type system $\vdash$ based on construct that describes the two ends of the same channel
  - An end point is described by session type qualified as linear or unrestricted
  - Linear types evolve to unrestricted types

- We assessed expressiveness by defining type-preserving encoding of
  1. linear lambda calculus [Walker&05]
  2. linear pi calculus [KPT TOPLAS'99]
  3. pi calculus with polarities [GH Acta'05]

Ongoing and future work

- We implemented rules $\vdash$ in type checking algorithm
- (Semantic) completeness in progress
- Still there are interesting processes that are not typable by $\vdash$

$$!x(y).(\nu a)(\overline{y}\langle a\rangle.a(\mathsf{title}).a(\mathsf{date}).(!a(\mathsf{date}) \mid \overline{a}\langle \mathsf{22March}\rangle))$$

- Both capabilities needed in continuation for receive and send date
- Sub typing à la Pierce&Sangiorgi would fix this