Subtyping and Typing Algorithms

for Mobile Ambients

Pascal ZIMMER

ENS Lyon / INRIA Sophia Antipolis

March 27, 2000

Introduction

The Ambient Calculus: A calculus designed to model mobility.

- An ambient is a bounded place in which computation happens.
- Ambients can be nested.
- An ambient can move into another one, out of another one or can be opened.

(Static) Type Systems: A “loose” specification, capable of detecting many programming errors.

Motivations

Cardelli, Ghelli and Gordon proposed a type system :

- to control the exchanges of values during communication
- to control the movement of ambients through other ambients
- to ensure that an ambient cannot be opened if it is locked

This type system is simple (no subtyping) and not algorithmic.

Our goal: to define a typing algorithm.

For this, we had to refine the types to allow subtyping.

Reduction I: Mobility

Entering an ambient:

$$\underbrace{n[\underline{in\ m}\ .\ P \mid Q]} \quad \mid \quad m[\underset{\Uparrow}{R}] \quad \rightarrow \quad m[n[P \mid Q] \mid R]$$

Exiting an ambient:

$$\Uparrow \quad m[\underbrace{n[\underline{out\ m}\ .\ P \mid Q]} \mid R] \quad \rightarrow \quad n[P \mid Q] \mid m[R]$$

Forcing a process to stay immobile:

$$\underline{imm}\ .\ P \ \rightarrow\ P$$

Reduction II: Open and Communication

Opening an ambient:

$$\underline{\mathit{open}\ n}\ .\ P \mid n[\ \underbrace{Q}\ ]\ _{\Uparrow} \quad \rightarrow \quad P \mid Q$$

Communicating a tuple of values:

$$\begin{aligned}(x_1 : W_1, \ldots, x_k : W_k)\ .\ P \mid \langle M_1, \ldots, M_k \rangle& \\ \rightarrow\ P\{x_1 \leftarrow M_1, \ldots, x_k \leftarrow M_k\}&\end{aligned}$$

Reduction III: Other Rules

$$P \;\rightarrow\; Q \;\Rightarrow\; (\nu n : W)P \;\rightarrow\; (\nu n : W)Q$$

$$P \;\rightarrow\; Q \;\Rightarrow\; n[P] \;\rightarrow\; n[Q]$$

$$P \;\rightarrow\; Q \;\Rightarrow\; P \mid R \;\rightarrow\; Q \mid R$$

$$P' \equiv P, P \;\rightarrow\; Q, Q \equiv Q' \;\Rightarrow\; P' \;\rightarrow\; Q'$$

where $\equiv$ is structural congruence (standard).

Syntax

$$
\begin{array}{rcll}
P, Q & ::= & (\nu n : \mathit{AmbType})P & \text{restriction} \\
 & & \mathbf{0} & \text{inactivity} \\
 & & P \mid Q & \text{composition} \\
 & & !P & \text{replication} \\
 & & M[P] & \text{ambient} \\
 & & M.P & \text{action} \\
 & & (x_1 : \mathit{MsgType}, \ldots, x_k : \mathit{MsgType}).P & \text{input} \\
 & & \langle M_1, \ldots, M_k \rangle & \text{async output} \\
 & & & \\
M & ::= & n \mid \mathit{in}\ M \mid \mathit{out}\ M \mid \mathit{open}\ M \mid \mathit{imm} & \text{expressions}
\end{array}
$$

The Original Process Types

$$T \quad ::= \quad {}^{Z}U$$

where: $Z \quad = \quad$ mobility annotation ($\curvearrowright$ or $\veebar$)

$U \quad = \quad$ Shh or type of values exchanged in the process

Examples:

- $\vdash \mathbf{0} \; : \; {}^{\veebar}Shh$ but also $\vdash \mathbf{0} \; : \; {}^{Z}U$ for any Z, U
- $\vdash \langle 1 \rangle \mid (x : Int).\mathbf{0} \; : \; {}^{\veebar}Int$
- $\vdash in\ n.\mathbf{0} \; : \; {}^{\curvearrowright}Shh$

But $\langle 1 \rangle \mid (x : Real).\mathbf{0}$ is not typable...

The New Process Types

$$T \quad ::= \quad {}^{Z}O \rightsquigarrow I$$

where:	Z	$=$	mobility annotation
	O	$=$	type of outputs (covariant)
	I	$=$	type of inputs (contravariant)

A communication can occur safely inside a process if $O \leq I$.

Examples:

- $\vdash \langle 1 \rangle \mid (x : Real).\mathbf{0} \;:\; {}^{Z}Int \rightsquigarrow Real$
- $\vdash \langle 1 \rangle \mid (x : Real).\mathbf{0} \;:\; {}^{Z}Real \rightsquigarrow Real$
- $\vdash \langle 1 \rangle \mid (x : Real).\mathbf{0} \;:\; {}^{Z}Int \rightsquigarrow Int$

The Original Input/Output Types

$$
\begin{array}{rcll}
U & ::= & Shh & \text{silence} \\
 & & W_1 \times \cdots \times W_k & \text{tuple}
\end{array}
$$

Defining a subtyping relation on tuples is easy. But what should we take for Shh ?

- A dumb process (which has no output) can be considered verbose. We should have $Shh \leq O$ for outputs.
- But a deaf process (which has no input) can also be considered hearing. We should have $I \leq Shh$ for inputs.
- So we should have $Shh \leq O \leq I \leq Shh$... (???)

The New Input/Output Types

$$
\begin{array}{lcll}
O, I & ::= & \bot & \text{bottom value} \\
 & & W_1 \times \cdots \times W_k & \text{tuple} \\
 & & \top & \text{top value}
\end{array}
$$

with $\bot < W_1 \times \cdots \times W_k < \top$

Value	Output term (O)	Input term (I)
$\bot$	No output Dumb process (Shh_{out})	There can be an input of any type
$\top$	There can be an output of any type	No input Deaf process (Shh_{in})

The New Input/Output Types

Examples:

- $Shh \approx \bot \rightsquigarrow \top$
- $\vdash \langle M_1 \rangle \mid \langle M_2 \rangle \ : \ {}^{Z}\top \rightsquigarrow \top$ with M_1 and M_2 incomparable (this process can be placed in parallel only with a deaf process)
- $\vdash (x : W_1).P \mid (y : W_2).Q \ : \ {}^{Z}\bot \rightsquigarrow \bot$ with W_1 and W_2 incomparable (this process can be placed in parallel only with a dumb process)

The Original Mobility Annotations

$$
\begin{array}{rcll}
Z & ::= & \curvearrowright & \text{mobile} \\
 & & \veebar & \text{immobile}
\end{array}
$$

What subtyping relation should we define ?

- If we privilege the generation of movement: $\veebar\ < \curvearrowright$
  (**0** is immobile but we should be able to consider it mobile in order to type $in\ n.\mathbf{0}$)
- If we privilege the restriction of immobile processes: $\curvearrowright < \ \veebar$
  (if $\veebar\ < \curvearrowright$, a process forced to stay immobile by typing can always remove this restriction with the subsumption rule)

The New Mobility Annotations

The solution: a 4-point lattice

$$Z ::= \begin{array}{ccccc} & & \overset{\rightarrow}{\top} & & \\ & \nearrow & & \nwarrow & \\ \underline{\vee} & & & & \curvearrowright \\ & \nwarrow & & \nearrow & \\ & & \underset{\rightarrow}{\bot} & & \end{array}$$

Examples:

- $\vdash \mathbf{0} \; : \; {}^{\underset{\rightarrow}{\bot}}\bot \leadsto \top$
- $\vdash in\ n.\mathbf{0} \; : \; {}^{\curvearrowright}\bot \leadsto \top$ and $\vdash imm.\mathbf{0} \; : \; {}^{\underline{\vee}}\bot \leadsto \top$
- $\vdash in\ n.\mathbf{0} \mid imm.\mathbf{0} \; : \; {}^{\overset{\rightarrow}{\top}}\bot \leadsto \top$
  $\Rightarrow$ a process type ${}^{Z}O \leadsto I$ is valid only if $Z \neq \overset{\rightarrow}{\top}$

Some Other Changes Needed

Locking annotations: $\bot_\circ < \bullet, \circ < \top^\circ$ instead of $\bullet, \circ$ (similar to the mobility annotations)

Ambient names: Originally, $Amb^Y[T]$ where T is exactly the type of processes inside the ambient. With subtyping, this is equivalent to allowing all processes of type $\leq T$. Thus, ambient names should be contravariant, but we get a problem with the *open* capability.

$\Rightarrow$ new ambient names: $Amb^Y[T, T']$ where

- T is the maximal type of processes allowed inside the ambient (contravariant)
- T' is the maximal effects if the ambient is opened (covariant)
- $T \leq T'$

The New Type System

We have written a new type system for the ambient calculus with subtyping.

Theorem 1 (Soundness) If a process is typable ($\vdash P : {}^{Z}O \rightsquigarrow I$), it cannot cause run-time errors ($O \leq I$ and $Z \neq \vec{\top}$).

Theorem 2 (Subject Reduction) If $E \vdash P : T$ and $P \ \rightarrow \ Q$, then $E \vdash Q : T$.

A First Typing Algorithm

We have defined a typing algorithm and showed its relation with the type system:

Theorem 3 (Minimal Type) The set of all possible types for a typable expression or process has a minimum and this minimum is precisely the type returned by the algorithm.

Theorem 4 (Subject Reduction) If $Type(E, P) = T$ and $P \rightarrow Q$, then the algorithm succeeds on Q and $Type(E, Q) \leq T$.

But this algorithm performs only type-checking...

A Type Inference Algorithm

- New and difficult problems arise (no minimal ambient type...)
- We considered the same problem for the original type system of Cardelli-Ghelli-Gordon.
- For this system, we could write a Damas-Milner style algorithm and show:

  Theorem 5 The type inference algorithm returns the most generic possible type.

- In our system with subtyping, still an open question...

Conclusion

Main benefits of our work:

- An extended type system with subtyping and a better comprehension of the underlying types
- A type checking algorithm
- A type inference algorithm for the original type system