

Semantic subtyping for the π -calculus

Giuseppe Castagna, Rocco de Nicola and **Daniele Varacca**
Paris, Firenze

Imperial College, November 5th 2004

Road Map

- Semantic subtyping
- Decidability
- The Semantic- π calculus
- Extensions

Road Map

- Semantic subtyping
 - Types for π
 - Channels as boxes
 - The intuitive semantic of types
 - The construction of a model
- Decidability
- The Semantic- π calculus
- Extensions

Typing channels in π

Channels are typed according to the value they transport

$$x : ch(t)$$

Processes are well typed when communication respects the typing

Well typed processes don't get stuck

Input/output channels

A more refined type system

$$x : ch^{-}(t)$$

The channel can be used only in output

$$x : ch^{+}(t)$$

The channel can be used only in input

Subtyping for π

Input types $ch^+(t)$:

what I expect to read from a channel

Covariant: channels carrying less don't harm

$$t \leq t' \implies ch^+(t) \leq ch^+(t')$$

Subtyping for π

Input types $ch^+(t)$:

what I expect to read from a channel

Covariant: channels carrying less don't harm

$$t \leq t' \implies ch^+(t) \leq ch^+(t')$$

Output types $ch^-(t)$:

what I expect I can write to a channel

Contravariant: channels carrying more don't harm

$$t \leq t' \implies ch^-(t) \geq ch^-(t')$$

Booleand combinators

What if we want unions, intersections, negations of types?

Semantic subtyping

The semantic of a type is the set of its values

$$\llbracket t \rrbracket = \{v \mid v : t\}$$

Subtyping is just set inclusion

Core of CDuce, a functional programming language
for XML

CDuce in one slide

Developped by V. Benzaken, G. Castagna, A. Frisch

- Based on Hosoya and Pierce's XDuce
- Native XML types
- Boolean combinators, function types, recursive types
- Semantic subtyping
- Pattern matching and typecase (based on subtyping)
- Efficiently implemented, and USED!

Foundational paper: *Semantic subtyping*, by Frisch, Castagna, Benzaken, LICS 2002

Our type system for π

Types $t ::= b \mid ch^+(t) \mid ch^-(t)$
 $\mid \mathbf{0} \mid \mathbf{1} \mid \neg t \mid t \vee t \mid t \wedge t$

Our type system for π

Types $t ::= b \mid ch^+(t) \mid ch^-(t)$
 $\mid \mathbf{0} \mid \mathbf{1} \mid \neg t \mid t \vee t \mid t \wedge t$
 $\mid t \times t \mid t \rightarrow t$
 $\mid X \mid \mathbf{rec} X.t$

Channel types

A channel is like a box with a particular shape
The box can contain only objects that fit that shape

$\llbracket ch(t) \rrbracket = \{ \text{channels that can contain objects of type } t \}$

Channel types

Assumption: types can only be distinguished by their shape

Channels **are** their shape

$$\llbracket ch(t) \rrbracket = \{\text{shape for objects of type } t\}$$

Channel types

Assumption: types can only be distinguished by their shape

Channels **are** their shape

$$\llbracket ch(t) \rrbracket = \{\text{set of objects of type } t\}$$

Channel types

Assumption: types can only be distinguished by their shape

Channels **are** their shape

$$\llbracket ch(t) \rrbracket = \{ \llbracket t \rrbracket \}$$

Invariance of channel types

Inupt types

Input/Output types express what can be safely done
If I am ready to read t on c , it is safe even if c carries less (gives me less)

$$\llbracket ch^+(t) \rrbracket = \{ \llbracket t' \rrbracket \mid t' \leq t \}$$

Covariance of input types

Output types

If I am ready to write t on c , it is safe even if c carries more (accepts more)

$$\llbracket ch^-(t) \rrbracket = \{ \llbracket t' \rrbracket \mid t' \geq t \}$$

Contravariance of output types

Boolean combinators

Boolean combinators are set-theoretical

$$\llbracket t \vee t' \rrbracket = \llbracket t \rrbracket \cup \llbracket t' \rrbracket$$

Some induced equations

$$ch^-(t) \wedge ch^+(t) = ch(t)$$

$$ch^+(t) \wedge ch^+(t') = ch^+(t \wedge t')$$

$$ch^+(t) \vee ch^+(t') \leq ch^+(t \vee t')$$

A Model for the types

A model is a set \mathcal{D} such that every type is interpreted as a subset of \mathcal{D}

Booleans interpreted as corresponding set-theoretic operations

Problem: denotations should be elements of \mathcal{D}

$$\llbracket ch^-(t) \rrbracket = \{ \llbracket t' \rrbracket \mid t' \geq t \}$$

I.e. subsets of \mathcal{D} should be elements of \mathcal{D}

Cantor is not happy

A model for the types

It is enough to build an interpretation that behaves **as if**

A model that induces the same subtyping as the intuitive semantics

A model for the types

A *Pre-Model* for the type is a function

$$\llbracket \cdot \rrbracket : \mathcal{T} \rightarrow \mathcal{P}(\mathcal{D})$$

that respects the boolean operators:

- $\llbracket \mathbf{1} \rrbracket = \mathcal{D}, \llbracket \mathbf{0} \rrbracket = \emptyset$
- $\llbracket \neg t \rrbracket = \mathcal{D} \setminus \llbracket t \rrbracket$
- $\llbracket t_1 \vee t_2 \rrbracket = \llbracket t_1 \rrbracket \cup \llbracket t_2 \rrbracket, \llbracket t_1 \wedge t_2 \rrbracket = \llbracket t_1 \rrbracket \cap \llbracket t_2 \rrbracket$

The extensional interpretation

The *extensional interpretation* is a function

$$\mathcal{E}[\] : \mathcal{T} \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{D}))$$

defined as:

- $\mathcal{E}[\mathbf{1}] = \mathcal{P}(\mathcal{D}), \mathcal{E}[\mathbf{0}] = \emptyset$
- $\mathcal{E}[\neg t] = \mathcal{E}[\mathbf{1}] \setminus \mathcal{E}[t]$
- $\mathcal{E}[t_1 \vee t_2] = \mathcal{E}[t_1] \cup \mathcal{E}[t_2],$
 $\mathcal{E}[t_1 \wedge t_2] = \mathcal{E}[t_1] \cap \mathcal{E}[t_2]$
- $\mathcal{E}[ch^+(t)] = \{\llbracket t' \rrbracket \mid \llbracket t \rrbracket' \subseteq \llbracket t \rrbracket\}$
- $\mathcal{E}[ch^-(t)] = \{\llbracket t' \rrbracket \mid \llbracket t \rrbracket \subseteq \llbracket t' \rrbracket\}$

The model condition

A Pre-model is a *Model*
if it behaves *as* the extensional interpretation

The model condition

A Pre-model is a *Model*
if it induces the same subtyping relation

The model condition

A Pre-model is a *Model*
if it induces the same subtyping relation

$$\llbracket t_1 \rrbracket \subseteq \llbracket t_2 \rrbracket$$

if and only if

$$\mathcal{E} \llbracket t_1 \rrbracket \subseteq \mathcal{E} \llbracket t_2 \rrbracket$$

Existence of a model

Does a model exist? Yes (a complicated construction)

We build a model in steps

For the $n + 1$ st step, a model of channel types
with $n + 1$ nesting
based on the n th step model

At the limit we have our \mathcal{D}

Important: not applicable to recursive types

Road Map

- Semantic subtyping
- Decidability
- The Semantic- π calculus
- Extensions

Road Map

- Semantic subtyping
- Decidability
 - Disjunctive normal form
 - The role of atoms
- The Semantic- π calculus
- Extensions

Decidability of subtyping

It is enough to decide emptiness:

$$t \leq t' \iff t \wedge \neg t' = \mathbf{0}$$

Decidability of subtyping

Put the type in disjunctive normal form

A disjunction is empty if all the summands are empty

A summand:

$$\bigwedge_{i \in P} t_i \wedge \bigwedge_{j \in N} \neg t'_j = \mathbf{0}?$$

Equivalently

$$\bigwedge_{i \in P} t_i \leq \bigvee_{j \in N} t'_j ?$$

Decidability of subtyping

Assume for basic type is decidable

Mixed basic and channel is easy

For channel types it reduces to

$$\bigwedge_{i \in I} ch^+(t_1^i) \wedge \bigwedge_{j \in J} ch^-(t_2^j) \leq \bigvee_{h \in H} ch^+(t_3^h) \vee \bigvee_{k \in K} ch^-(t_4^k)$$

Decidability of subtyping

Assume for basic type is decidable

Mixed basic and channel is easy

For channel types it reduces to

$$ch^+(t_1) \wedge ch^-(t_2) \leq \bigvee_{h \in H} ch^+(t_3^h) \vee \bigvee_{k \in K} ch^-(t_4^k)$$

Atoms

The condition to check involves **atoms**

Types with a singleton interpretation

Decide whether a type is an atom

Decide whether a type is finite

Example of subtyping

Three constants $\text{err}_1, \text{err}_2, \text{exc}$

$$t_2 = \text{int}$$

$$t_1 = t_2 \vee \text{err}_1 \vee \text{err}_2 \vee \text{exc}$$

$$t_3 = t_2 \vee \text{exc}$$

$$t_4 = t_2 \vee \text{err}_1 \vee \text{err}_2$$

$$ch^+(t_1) \wedge ch^-(t_2) \not\leq ch^+(t_3) \vee ch^-(t_4)$$

Example of subtyping

Three constants $\text{err}_1, \text{err}_2, \text{exc}$

$$t_2 = \text{int}$$

$$t_1 = t_2 \vee \text{err}_1 \vee \text{err}_2 \vee \text{exc}$$

$$t_3 = t_2 \vee \text{exc}$$

$$t_4 = t_2 \vee \text{err}_1 \vee \text{err}_2$$

$$ch^+(t_1) \wedge ch^-(t_2) \leq ch^+(t_3) \vee ch^-(t_4)$$

if $\text{err}_1 = \text{err}_2$

Example of subtyping

Three constants $\text{err}_1, \text{err}_2, \text{exc}$

$$t_2 = \text{int}$$

$$t_1 = t_2 \vee \text{err}_1 \vee \text{err}_2 \vee \text{exc}$$

$$t_3 = t_2 \vee \text{exc}$$

$$t_4 = t_2 \vee \text{err}_1 \vee \text{err}_2$$

it depends whether $\text{err}_1 \vee \text{err}_2$ is an atom

Relevance of atomic types

Atomic types important also in presence of polymorphism

Hosoya, Frisch, Castagna *Parametric Polymorphism for XML*, POPL 2005

Try to avoid atomic types!

Road Map

- Semantic subtyping
- Decidability
- The Semantic- π calculus
- Extensions

Road Map

- Semantic subtyping
- Decidability
- The Semantic- π calculus
 - Patterns
 - The language
 - Typing
 - Operational semantics
- Extensions

Patterns

$p ::= x$ capture, $x \in \mathbb{V}$

| $p \wedge t$ conjunction

| $p_1 | p_2$ alternative

in $p_1 | p_2$ we have $Var(p_1) = Var(p_2)$

Pattern matching

Matching an element of the **model**

$$d/x = \{x \mapsto d\}$$

$$d/p \wedge t = d/p \quad \text{if } d \in \llbracket t \rrbracket$$

$$d/p_1 | p_2 = d/p_1 \quad \text{if } d/p_1 \text{ does not fail}$$

$$d/p_1 | p_2 = d/p_2 \quad \text{if } d/p_1 \text{ fails}$$

Semantic- π

<i>Channels</i>	α	$::=$	x	variables
			$ $	
			c^t	typed channel (box)
<i>Messages</i>	M	$::=$	n	constant
			$ $	
			α	channel
<i>Processes</i>	P	$::=$	$\bar{\alpha}M$	output
			$ $	
			$\sum_{i \in I} \alpha(p_i).P_i$	patterned input
			$ $	
			$P_1 \parallel P_2$	parallel
			$ $	
			$(\nu c^t)P$	restriction
			$ $	
			$!P$	replication

Distinctive features

- Asynchrony
- Pattern Matching
- Distinction between variables and constants (as in Nielsen & Engberg ECCS)
- Channels are explicitly typed

Typing Messages

$$\overline{\Gamma \vdash n : b_n} \text{ (const)}$$

$$\overline{\Gamma \vdash c^t : ch(t)} \text{ (chan)}$$

$$\overline{\Gamma \vdash x : \Gamma(x)} \text{ (var)}$$

$$\frac{\Gamma \vdash M : s \leq t}{\Gamma \vdash M : t} \text{ (subs)}$$

Typing processes

$$\frac{\Gamma \vdash P}{\Gamma \vdash (\nu c^t)P} \text{ (new)} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P} \text{ (repl)}$$

$$\frac{\Gamma \vdash P_1 \quad \Gamma \vdash P_2}{\Gamma \vdash P_1 \parallel P_2} \text{ (para)}$$

$$\frac{t \leq \bigvee_{i \in I} \wp p_i \quad \bigwedge_{p_i} \wp p_i \wedge t \neq \mathbf{0} \quad \frac{\Gamma \vdash \alpha : ch^+(t) \quad \Gamma, t/p_i \vdash P_i}{\Gamma \vdash \sum_{i \in I} \alpha(p_i).P_i} \text{ (input)}}{\Gamma \vdash \sum_{i \in I} \alpha(p_i).P_i} \text{ (input)}$$

$$\frac{\Gamma \vdash M : t \quad \Gamma \vdash \alpha : ch^-(t)}{\Gamma \vdash \bar{\alpha}M} \text{ (output)}$$

The model of values

New semantics of types

$$\llbracket t \rrbracket = \{v \mid \vdash v : t\}$$

It induces the same subtyping relation

It allows the pattern matching on values

The operational semantics

$$\overline{c^t v} \parallel \sum_{i \in I} c^t(p_i).P_i \longrightarrow P_j[v/p_j]$$

“Call by value”

Subject reduction

Decidability of typing

Decidability of subtyping does not entail decidability of typing!

Need to find principal types

This possible!

Road Map

- Semantic subtyping
- Decidability
- The Semantic- π calculus
- Extensions

Road Map

- Semantic subtyping
- Decidability
- The Semantic- π calculus
- Extensions
 - Product and arrow types
 - Recursive types

Products

Type system and language can be extended with product types

A polyadic version

The subtyping is still decidable

Recursive types

Only a limited form of recursive type:

$$t = \text{int} \vee (ch(t) \wedge ch(\text{int}))$$

Question: is $t = \text{int}$ or not?

Both lead to contradiction

Recursive types

A recursion variable cannot occur inside a channel type

We can still type a channel carrying itself using $ch(\mathbf{1})$

We can express the type of list

The model is much more complicated (CDuce)

Decidability? May be yes...

Arrow types

Arrow types: passing higher order functions
(\mathbb{C} Duce values)

Weak version: passing \mathbb{C} Duce functions that cannot
act on channels

Strong version: \mathbb{C} Duce functions using channels

Decidability? Undecidability?

Examples

A server

$$\alpha(x).!(x(y \wedge t_1).P_1 + x(y \wedge t_2).P_2)$$

$$\alpha : ch^+(ch^+(t_1 \vee t_2))$$

Examples

A server

$$\alpha(x).!(x(y \wedge t_1).P_1 + x(y \wedge t_2).P_2)$$

$$\alpha : ch^+(ch^+(t_1 \vee t_2))$$

A more efficient server

$$\alpha(x \wedge ch(t_1)).!x(y).P_1 + \alpha(x \wedge ch(t_2)).!x(y).P_2$$

$$\alpha : ch^+(ch^+(t_1) \vee ch^+(t_2))$$

Examples

A CDuce server

$$\mathit{fun}^{s \rightarrow t}(x).\mathit{arg}^s(y).\overline{\mathit{result}}^t(x(y))$$

A more liberal one

$$\begin{aligned} & \mathit{compute}^{((s \rightarrow t) \times s) \vee (s \times (s \rightarrow t))}(x, y \wedge s).\overline{\mathit{result}}^t(x(y)) \\ + & \mathit{compute}^{((s \rightarrow t) \times s) \vee (s \times (s \rightarrow t))}(x \wedge s, y).\overline{\mathit{result}}^t(y(x)) \end{aligned}$$

Examples

A $\mathbb{C}\pi$ function

```
fun assoc(s : string , l : a_list) : ch(int) =  
  match l with nil → fail  
    | ((k,c),t) → if k = s then c  
                      else assoc(s,t)
```

A $\mathbb{C}\pi$ process

```
announce[m_list × a_list × retr] (marks, mails, getch).  
  (νcm_list)  $\bar{c}$ (marks) |  
    !( c( ((n,m),rest) ) . (  $\overline{\text{getch}(\textit{n}, \textit{mails})}(\textit{m})$  |  $\bar{c}(\textit{rest})$  )  
      + c( nil ). 0 )
```

$m_list = ((string \times int) \times m_list) \vee nil$

$a_list = ((string \times ch(int)) \times a_list) \vee nil$

$retr = string \times a_list \rightarrow ch(int)$

Examples

$$\overline{\text{announce}}((("Alice", 6), (("Bob", 8), \text{nil})), \\ (("Bob", c_B), (("Alice", c_A), \text{nil})), \\ \text{assoc})$$

Will reduce to process: $\overline{c_A}(6) \mid \overline{c_B}(8)$.

Conclusion

- π -calculus with pattern matching
- a very rich type system
- advantages of semantic subtyping
- integration with \mathbb{C} Duce, a functional programming language for XML documents:
 $\mathbb{C}\pi$ -calculus

Any questions?