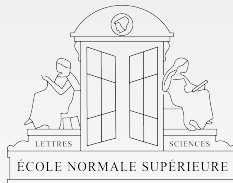


TAPIS

Mehdi DOGGUY

PARKAS team
Laboratoire d'Informatique de l'École Normale Supérieure, Paris



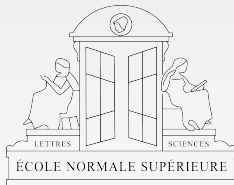
November 2011

Synchron 2011, Le Bois du Lys

Timed Asynchronous π -calculus

Mehdi DOGGUY

PARKAS team
Laboratoire d'Informatique de l'École Normale Supérieure, Paris



November 2011

Synchron 2011, Le Bois du Lys

Introduction

- ▶ Synchronous π -calculus

- ▶ Time based interactions: you can do things today **or** *maybe* tomorrow

$$s(x).P, K$$

- ▶ Asynchronous communications based on signals:
 - ▶ the sender doesn't need to wait for a receiver to get the message out of the channel
 - ▶ there is a hidden recursion when using signals
 - ▶ Can use recursive threads
 - ▶ Can dereference signals at the end of the instant
- ▶ R. Amadio. [A synchronous \$\pi\$ -calculus](#). Information and Computation, 205(9):1470–1490, 2007.
- ▶ R. Amadio and M. Dogguy. [Determinacy in a synchronous pi-calculus](#). 'From semantics to computer science: essays in honor of Gilles Kahn', Cambridge University Press, 2008.

Goals

- ▶ We want a simpler framework, but with rich interactions
 - ▶ channel based communication and name passing
 - ▶ no hidden recursions
 - ▶ no recursive threads
 - ▶ no dereferencing at the end of the instant
- ▶ Proof of determinacy (by means of confluence)

2 TAPIS

Syntax

Semantics

Examples

Asynchronous π -calculus

Timed CCS



TAPIS

time passes exactly when no computation is possible

- ▶ R. Milner, J. Parrow, and D. Walker. [A calculus of mobile processes, parts 1-2](#). *Information and Computation*, 100(1):1–77, 1992.
- ▶ Yi, Wang. [Real-time behaviour of asynchronous agents](#). In proc. *CONCUR '90*, Springer 458, pages 502–520, 1990.
- ▶ Roberto Amadio, Mehdi Dogguy. [On Affine Usages in Signal-Based Communication](#). APLAS 2008: 221–236
- ▶ Roberto Amadio, [On convergence sensitive bisimulation and the embedding of CCS in timed CCS](#). Electronic Notes in TCS 242. 2009.

Asynchronous π -calculus

Timed CCS

TAPIS

$$P \triangleright Q$$

$$\frac{P \not\rightarrow \cdot}{P \triangleright Q \xrightarrow{\text{tick}} Q}$$

time passes exactly when no computation is possible

- ▶ R. Milner, J. Parrow, and D. Walker. [A calculus of mobile processes, parts 1-2](#). *Information and Computation*, 100(1):1–77, 1992.
- ▶ Yi, Wang. [Real-time behaviour of asynchronous agents](#). In proc. *CONCUR '90*, Springer 458, pages 502–520, 1990.
- ▶ Roberto Amadio, Mehdi Dogguy. [On Affine Usages in Signal-Based Communication](#). APLAS 2008: 221–236
- ▶ Roberto Amadio, [On convergence sensitive bisimulation and the embedding of CCS in timed CCS](#). Electronic Notes in TCS 242. 2009.

Syntax

Programs $P, Q, \dots$ in TAPIS are defined as follows:

Processes

P, Q	$::=$	0	terminated process
	$ $	$P \mid Q$	parallel composition
	$ $	$\nu a \, P$	name creation
	$ $	$[t \sqsubseteq p]P, Q$	pattern matching
	$ $	R	

Actions

R	$::=$	$\bar{a} \langle t \rangle$	output
	$ $	$a(x).P$	input
	$ $	$a(x).P \triangleright Q$	timed input
	$ $	$!a(x).P$	replicated input
	$ $	$a(x).P \,! \triangleright Q$	replicated timed input

It turns out that $\cdot !\triangleright \cdot$ is definable using other operators:

$$a(x).P !\triangleright Q \stackrel{def}{=} !A.(\textcolor{blue}{a}(x).(P \mid \overline{A}<>), 0) \\ \mid \textit{pause}.Q \\ \mid \overline{A}<>$$

where $\textit{pause}.P = \nu a (a(x).0 \triangleright P)$.

... but this encoding is not satisfactory.

Semantics (during the instant)

$$(comp) \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$$

$$(\nu) \frac{P \rightarrow P'}{\nu a \, P \rightarrow \nu a \, P'}$$

$$(=^ind_1) \frac{match(t, p) = \theta}{[t \sqsubseteq p] P_1, P_2 \rightarrow \theta P_1}$$

$$(=^ind_2) \frac{match(t, p) = \uparrow}{[t \sqsubseteq p] P_1, P_2 \rightarrow P_2}$$

$$(\equiv) \frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}$$

$$(sync) \frac{}{E[a(x).P] \mid \bar{a} < t > \rightarrow [t/x]P}$$

$$(!) \frac{}{F[a(x).P] \mid \bar{a} < t > \longrightarrow F[a(x).P] \mid [t/x]P}$$

where:

$$\begin{array}{lcl} E & ::= & [] \quad \parallel \quad [] \triangleright P \\ F & ::= & ![] \quad \parallel \quad [] !\triangleright P \end{array}$$

Semantics (end of the instant)

$$(\triangleright_{else}) \frac{}{a(x).P \triangleright Q \xrightarrow{tick} Q} \quad (\triangleright_{in}) \frac{}{a(x).P \xrightarrow{tick} a(x).P}$$

$$(\triangleright_{rec}) \frac{}{a(x).P !\triangleright Q \xrightarrow{tick} Q} \quad (\triangleright_{!}) \frac{}{!a(x).P \xrightarrow{tick} !a(x).P}$$

$$(\triangleright_{\nu}) \frac{P \xrightarrow{tick} P'}{\nu a P \xrightarrow{tick} \nu a P'} \quad (\triangleright_{out}) \frac{}{\bar{a} \langle t \rangle \xrightarrow{tick} 0}$$

$$(\triangleright_{comp}) \frac{P \xrightarrow{tick} P' \quad Q \xrightarrow{tick} Q' \quad P \mid Q \not\rightarrow .}{P \mid Q \xrightarrow{tick} P' \mid Q'}$$

Examples (1/2)

Signals defined with channels:

$$\text{emit}(a, v) = a(x).\bar{x}\langle v \rangle ! \triangleright 0$$

$$\text{present } a(x) \text{ do } P \text{ else } Q = \nu r (\bar{a}\langle r \rangle \mid r(x).P \triangleright Q)$$

Examples (2/2)

Recursive threads:

let $\text{rec } A(x) = P \text{ in } Q$ can be coded as follows:

$$!A(x).[\bar{A}\langle x \rangle / A(x)]P \mid [\bar{A}\langle x \rangle / A(x)]Q$$

3 Typing system

Usages

Types

Typing rules

Properties

Examples

Usages

Usages are:

$$u \in L = \left\{ \begin{array}{ll} 0 & \text{Cannot be used} \\ 1 & \text{Used at most once} \\ \infty & \text{Used an arbitrary number of times} \end{array} \right\}$$

Partial sum¹ operation on usages

$$u \oplus u' = \left\{ \begin{array}{ll} 0 \oplus x = x & x \in L \\ 1 \oplus 1 = \uparrow \\ \infty \oplus 1 = \uparrow \\ \infty \oplus x = \infty & x \in \{0, \infty\} \end{array} \right.$$

¹up to symmetry

Channel's usage

$$s : Ch_{\binom{n}{m}}^{\omega}(\sigma)$$

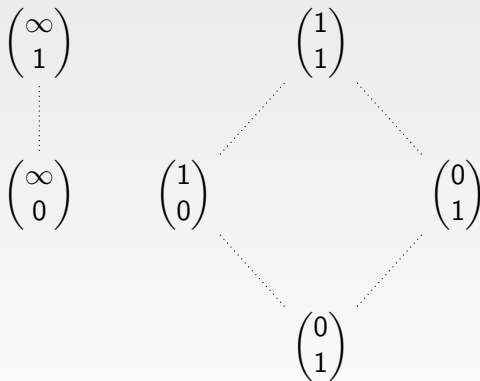
where: $\begin{cases} n & \text{maximum number of outputs} \\ m & \text{maximum number of inputs} \end{cases}$

$$n, m \in L = \{0, 1, \infty\}$$

Addition over channel usages

$$Ch_u(\sigma) \oplus Ch_v(\sigma) = Ch_{u \oplus v}(\sigma)$$

Usages: classification



Types

$$\kappa ::= T_{\infty}(\kappa_1, \dots, \kappa_n) \mid Ch_u(\kappa)$$

Classical uniform types
(*u is neutral*)

Lift operation on types

$$\begin{aligned}\uparrow Ch_{u \cdot v^{\omega}}(\sigma) &= Ch_{v^{\omega}}(\sigma) \\ \uparrow T_u(\sigma_1, \dots, \sigma_n) &= T_u(\sigma_1, \dots, \sigma_n)\end{aligned}$$

Types

$$\kappa ::= T_{\infty}(\kappa_1, \dots, \kappa_n) \mid Ch_u(\kappa)$$

Classical uniform types
(*u is neutral*)

$$\lambda ::= T_1(\sigma_1, \dots, \sigma_n) \mid Ch_v(\sigma)$$

Affine types
(*v is affine*)

Lift operation on types

$$\begin{aligned}\uparrow Ch_{u \cdot v^{\omega}}(\sigma) &= Ch_{v^{\omega}}(\sigma) \\ \uparrow T_u(\sigma_1, \dots, \sigma_n) &= T_u(\sigma_1, \dots, \sigma_n)\end{aligned}$$

Types

$$\kappa ::= T_{\infty}(\kappa_1, \dots, \kappa_n) \mid Ch_u(\kappa)$$

Classical uniform types
(*u is neutral*)

$$\lambda ::= T_1(\sigma_1, \dots, \sigma_n) \mid Ch_v(\sigma)$$

Affine types
(*v is affine*)

$$\sigma ::= \kappa \mid \lambda$$

Types

Lift operation on types

$$\begin{aligned}\uparrow Ch_{u \cdot v^{\omega}}(\sigma) &= Ch_{v^{\omega}}(\sigma) \\ \uparrow T_u(\sigma_1, \dots, \sigma_n) &= T_u(\sigma_1, \dots, \sigma_n)\end{aligned}$$

Typing rules

$$(0) \frac{}{\Gamma \vdash 0} \qquad (par) \frac{\Gamma_i \vdash P_i \quad i = 1, 2}{\Gamma_1 \oplus \Gamma_2 \vdash P_1 \mid P_2}$$

$$(out) \frac{\Gamma_a \vdash a : Ch_{u.v\omega}(\sigma) \quad \Gamma_e \vdash e : \sigma \quad u \in \{(1, 0), (\infty, 0)\}}{\Gamma_a \oplus \Gamma_e \vdash \bar{a} \langle e \rangle}$$

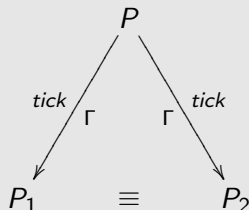
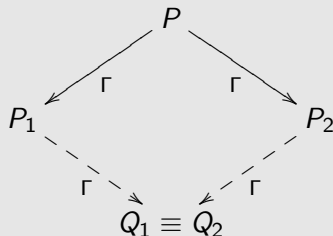
$$(! \triangleright in) \frac{\Gamma_a \vdash a : Ch_{u.v\omega}(\sigma) \quad \Gamma_P, x : \sigma \vdash P \quad u = (\infty, 1) \quad \uparrow \Gamma_Q \vdash Q \quad neutral(\Gamma_P)}{\Gamma_a \oplus \Gamma_P \oplus \Gamma_Q \vdash a(x).P !\triangleright Q}$$

Properties

Lemma (Subject Reduction)

- ▶ If $\Gamma \vdash P$ and $P \longrightarrow P'$ then $\Gamma \vdash P'$
- ▶ If $\Gamma \vdash P$ and $P \xrightarrow{\text{tick}} P'$ then $\uparrow \Gamma \vdash P'$

Lemma (Confluence)



Properties (Subject reduction in Coq)

Lemma (Subject Reduction in Coq)

*Lemma subject_reduction_1 : forall P P' Γ ,
 $\Gamma \vdash P \rightarrow P \longrightarrow P' \rightarrow \Gamma \vdash P$.*

*Lemma subject_reduction_2 : forall P P' Γ ,
 $\Gamma \vdash P \rightarrow P \xrightarrow{\text{tick}} P' \rightarrow \uparrow \Gamma \vdash P$.*

```
$ wc -l *.v
    343  CoreLibrary.v
    743  Syntax.v
   2109  Additions.v
   2719  Names.v
   3014  Typing.v
   3071  Tapis.v

11 999  total
```

Examples (1/2)

Signals defined with channels:

$$\begin{aligned} \nu : \sigma, a : Ch_{(\infty,1)\omega}(\sigma_r) \vdash & \text{emit}(a, \nu) \\ & = a(x).\bar{x}\langle \nu \rangle ! \triangleright 0 \end{aligned}$$

$$\begin{aligned} \Gamma_P \oplus \Gamma_Q \oplus a : Ch_{(1,1)\omega}(\sigma_r \oplus \sigma_R) \vdash & \text{present } a(x) \text{ do } P \text{ else } Q \\ & = \nu r (\bar{a}\langle r \rangle \mid r(x).P \triangleright Q) \end{aligned}$$

where $\sigma_r = Ch_{(1,0).(0,0)\omega}(\sigma)$ and $\sigma_R = Ch_{(0,1).(1,1)\omega}(\sigma)$

Examples (2/2)

Recursive (eventually, mutually defined) threads:

let rec $A(x) = P$ in Q can be coded as follows:

$$!A(x).[\overline{A}\langle x \rangle / A(x)]P \mid [\overline{A}\langle x \rangle / A(x)]Q$$

typable in $\Gamma = A : Ch_{(\infty,1)\omega}(\sigma) \oplus \Gamma_P \oplus \Gamma_Q$

4 Conclusion

Conclusion

- ▶ TAPIS is nice calculus² in which we can encode several other known calculi
- ▶ Usages are used by the typing system to avoid chaos
- ▶ Proof of subject reduction lemma is formalized in Coq

²Believe me!

Conclusion

- ▶ TAPIS is nice calculus² in which we can encode several other known calculi
- ▶ Usages are used by the typing system to avoid chaos
- ▶ Proof of subject reduction lemma is formalized in Coq

What's next?

- ▶ Extract a certified type checker for TAPIS
- ▶ Proof of confluence in Coq

²Believe me!