Processes As Coalgebras

Luís S. Barbosa

UNIVERSITY OF TARTU

December 2003

Departamento de Informática — Universidade do Minho

Motivation

- Initial algebras and final coalgebras, as universal objects, entail a basis for program development directly based on (actually driven by) type specifications, providing

	Definition	*Proof*
Initial algebras	recursion	induction
final coalgebras	co-recursion	co-induction

- The role of such universals has been fundamental to a whole discipline of algorithm transformation (the Bird-Meertens style).
- Moreover, such properties can be turned into programming combinators and used, not only to calculate programs, but also to program with.

Aims

Apply identical reasoning principles and calculational style to the design of process calculi
... relying on the representation of processes as inhabitants of final coalgebras for suitable Set endofunctors

- Uniform treatment of processes and data
- Direct derivation of functional interpreters for process calculi
- Genericity: structural aspects of the underlying behaviour model clearly separated from the interaction structure which defines the synchronisation discipline
- Proofs in a calculational (basically equational and pointfree) style

The Design Problem

Operational semantics given in terms of an action-indexed transition relation $\xrightarrow{a}$ over processes

... witnessing the collection of actions in which a process gets committed and the resulting 'continuations'

$\Rightarrow$ design decisions:

- 'collection of' $\leadsto$ **behaviour model** B
- 'transition relation' $p \xrightarrow{a} q$ admits different interpretations, *e.g.*,
  - *Active*: 'p evolves to q by performing an action a, both q and a being solely determined by p'
  - *Reactive*: 'p reacts to an external stimulus a by evolving to q'

  $\leadsto$ different **shapes**
- 'actions' $\leadsto$ a variety of **interaction disciplines**

The Approach

- Represent processes as inhabitants of the carrier of the final coalgebra for suitable Set functors

$$\omega : \nu \longrightarrow \mathsf{T}\ \nu$$

- so that definition of process' combinators and proofs of their properties rely on finality
- which entails direct derivations of functional interpreters for (classical) process calculi

The Approach

- Let ν_{T} denote (the) final universe of processes of shape T
- and $\square$ a process combinator such that

$$\begin{tikzcd} \nu_{\mathsf{T}} \arrow[r, "\omega_{\mathsf{T}}"] & \mathsf{T}\ \nu_{\mathsf{T}} \\ U \arrow[u, "\square"] \arrow[r, "\alpha_{\square}"] & \mathsf{T}\ U \arrow[u, "\mathsf{T}\ \square"'] \end{tikzcd}$$

commutes for a gene coalgebra $\alpha_{\square}$ which specifies its one-step dynamics, i.e.

$$\square \quad = \quad [\![\alpha_{\square}]\!]_{\mathsf{T}}$$

said its anamorphism or coinductive extension

Process Calculi

Adopt

$$\boxed{\mathsf{T} \;=\; \mathcal{P}(Act \times \mathsf{Id})}$$

- corresponding to a active interpretation: processes are elements of the final coalgebra $\omega : \nu \longrightarrow \mathcal{P}(Act \times \nu)$ and the transition relation is defined as

$$p \xrightarrow{a} q \quad \text{iff} \quad \langle a, q \rangle \in \omega\ p$$

- but restricting to image-finite processes

Process Calculi

Declaration

```
data C -> Pr(A) = bh:  C -> set(A * C).
```

plus

- a definition of the interaction discipline
- and dynamic and static process combinators
- ... parametric on such a discipline

Process Calculi

Dynamic Combinators

- consumed on action occurrence: disappear from the expression representing the process continuation.

$$
\begin{array}{rrcl}
\text{inaction} & \omega \cdot \mathsf{nil} & = & \underline{\emptyset} \\
\text{prefix} & \omega \cdot a. & = & \mathsf{sing} \cdot \mathsf{label}_a \\
\text{choice} & \omega \cdot + & = & \cup \cdot (\omega \times \omega)
\end{array}
$$

where $\mathsf{sing} = \lambda x.\ \{x\}$ and $\mathsf{label}_a = \lambda x.\ \langle a, x \rangle$.

Process Calculi

Laws

$\langle \nu; +, \mathsf{nil} \rangle$ is an Abelian idempotent monoid

$$
\begin{array}{rcl}
(p+q)+r = p+(q+r) & \equiv & + \cdot (+ \times \mathsf{id}) = + \cdot (\mathsf{id} \times +) \cdot \mathsf{a} \\
p + \mathsf{nil} = p & \equiv & + \cdot (\mathsf{id} \times \mathsf{nil}) \cdot \mathsf{l}^\circ = \mathsf{id} \\
p + p = p & \equiv & + \cdot \triangle = \mathsf{id} \\
p + q = q + p & \equiv & + \cdot \mathsf{s} = +
\end{array}
$$

Process Calculi

Simple Equational Proof: $+ \cdot \mathsf{s} = +$

$$
\begin{aligned}
& \omega \cdot + \cdot \mathsf{s} \\
\equiv \quad & \{ \text{ definition } \} \\
& \cup \cdot (\omega \times \omega) \cdot \mathsf{s} \\
\equiv \quad & \{ \text{ s natural } \} \\
& \cup \cdot \mathsf{s} \cdot (\omega \times \omega) \\
\equiv \quad & \{ \cup \text{ commutative } \} \\
& \cup \cdot (\omega \times \omega) \\
\equiv \quad & \{ \text{ definition } \} \\
& \omega \cdot +
\end{aligned}
$$

Process Calculi

```
def bnil: 1 -> Pr(A)
    = () => (bh: empty).


def bpre: A * Pr(A) -> Pr(A)
    = (a, t)  => (bh: sing(a,t)).


def bcho: Pr(A) * Pr(A)  -> Pr(A)
    = (t1, t2) => (bh: union(bh t1, bh t2)).
```

Process Calculi

Static Combinators

- Persistence through action occurrence enforces (co)recursive definition
- Therefore, arise as anamorphisms generated by suitable 'gene' coalgebras
- In general, depend on the calculus interaction structure

Process Calculi

Interaction Structure

$\langle Act; \theta, 1 \rangle$ abelian positive monoid with a zero 0

- θ determines the interaction discipline
- 0 represents the absence of interaction: for all $a \in Act$, $a \theta 0 = 0$.
- 1 technical role to equip the behaviour functor with a monadic structure
- being positive means $a \theta a' = 1$ iff $a = a' = 1$

Process Calculi

Interaction Structure: Examples

- Co-occurrence: $a\theta b = \langle a, b \rangle$, for all $a, b \in Act$ $(\neq 0, 1)$

parametric over a set L of labels and encoded as

```
data Ac(L) -> A = act: L -> A | syn: A * A -> A
               | nop: 1 -> A | idle: 1 -> A.

def prodAc : Ac(L) * Ac(L) -> Ac(L)
  = (nop, - ) => nop | ( - ,nop) => nop
  | (idle, x) => x   | (x ,idle) => x
  | ( a1, a2) => syn( a1, a2).
```


Process Calculi

Interaction Structure: Examples

- CCS [Milner89]:
  - L has an involution ($\overline{a}$, for $a \in L$)
  - $a\theta\overline{a} = \tau$ $\quad$ $a\theta 1 = a$ $\quad$ $a\theta b = 0$, for $b \neq \overline{a}$
- CSP [Hoare85]:

$$a\theta a \;=\; a\,, \quad a\theta 1 \;=\; a \;\text{ and }\; a\theta b \;=\; 0 \;\text{ in all other cases}$$

- ...

Process Calculi

Renaming

Once fixed an interaction structure any homomorphism

$$f : Act \longrightarrow Act$$

lifts to a renaming combinator

$$[f] \;=\; [\![\alpha_{[f]}]\!]$$

defined as the coinductive extension of

$$\alpha_{[f]} \;=\; \nu \xrightarrow{\omega} \mathcal{P}(Act \times \nu) \xrightarrow{\mathcal{P}(f \times \mathsf{id})} \mathcal{P}(Act \times \nu)$$

Process Calculi

Renaming

$$\begin{array}{ccc} \nu & \xrightarrow{\qquad\qquad\omega\qquad\qquad} & \mathcal{P}(Act \times \nu) \\ \uparrow{\scriptstyle [f]} & & \uparrow{\scriptstyle \mathcal{P}([f] \times \mathsf{id})} \\ \nu \xrightarrow{\omega} \mathcal{P}(Act \times \nu) & \xrightarrow{\mathcal{P}(f \times \mathsf{id})} & \mathcal{P}(Act \times \nu) \end{array}$$

Process Calculi

Encoding

```
def bren{eqL: Lb(N) * Lb(N) -> bool}:
     Pr(Ac(Lb(N))) * map(Ac(Lb(N)),Ac(Lb(N)))
     -> Pr(Ac(Lb(N)))

 = (t, h) =>
   (| r => bh: set{x =>
        { ff => x | ss a  => (a, p1 x)
        } app{eqA{eqL}}(compren h, p0 x) }  (bh r)
    |) t.
```


Process Calculi

Restriction

$$\backslash_K = [\![\alpha_{\backslash_K}]\!]$$

i.e., the coinductive extension of

$$\alpha_{\backslash_K} \;=\; \nu \xrightarrow{\omega} \mathcal{P}(A \times \nu) \xrightarrow{\mathsf{filter}_K} \mathcal{P}(A \times \nu)$$

where $\mathsf{filter}_K = \lambda s \,.\, \{t \in s |\, \pi_1\, t \notin K\}$

Process Calculi

Encoding

```
def bret{eqL: Lb(N) * Lb(N) -> bool}:
        Pr(Ac(Lb(N))) * set(Lb(N)) -> Pr(Ac(Lb(N)))

 = (t, k) =>
   (| r => bh:
      filter{x => not member{eqA{eqL}}
            (p0 x, compret k)} (bh r) |) t.
```

where `compret` closes K by the label structure, *e.g.*, in Ccs

```
def compret: set(Lb(N)) -> set(Ac(Lb(N)))
 = s => union( set{l => act(l)} s ,
               set{l => act(inv(l)) } s).
```


Process Calculi

Interleaving

$$\mathbin{|||} \;=\; [\![\alpha_{|||}]\!]$$

defined as the coinductive extension of

$$\alpha_{|||} \;=\; \nu \times \nu \xrightarrow{\triangle} (\nu \times \nu) \times (\nu \times \nu)$$

$$\xrightarrow{(\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)} (\mathcal{P}(Act \times \nu) \times \nu) \times (\nu \times \mathcal{P}(Act \times \nu))$$

$$\xrightarrow{\tau_r \times \tau_l} \mathcal{P}(Act \times (\nu \times \nu)) \times \mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\cup} \mathcal{P}(Act \times (\nu \times \nu))$$

Process Calculi

Encoding

```
def bint: Pr(Ac(L)) * Pr(Ac(L))  -> Pr(Ac(L))
 = (t1, t2) =>
   (| (r1,r2) =>
        bh: union(taur(bh r1, r2), taul(bh r2, r1))
    |) (t1,t2).
```

where strength is encoded as

```
def taur = (s,t) => set {(a,x) => (a, (x,t))} s.
```

Laws: $\langle |||, \mathsf{nil}\rangle$ is an Abelian monoid

Process Calculi

Laws: restriction and renaming

$$\setminus_K \cdot + = + \cdot (\setminus_K \times \setminus_K)$$
$$\setminus_K \cdot ||| = ||| \cdot (\setminus_K \times \setminus_K)$$
$$\setminus_K \cdot a. = (a \in K \;\to\; \mathsf{nil},\, a. \cdot \setminus_K)$$
$$\setminus_K \cdot \setminus_K = \setminus_K$$

$$\begin{aligned} [\mathsf{id}] &= \mathsf{id} \\ [f] \cdot [f'] &= [f \cdot f'] \\ [f] \cdot a. &= (f\ a). \cdot [f] \\ [f] \cdot + &= + \cdot ([f] \times [f]) \\ [f] \cdot ||| &= ||| \cdot ([f] \times [f]) \end{aligned}$$

Proofs and Calculation

Proofs by Fusion: $\mid\mid\mid$ commutativity

By definition

$$||| \cdot \mathsf{s} = |||$$

equivales

$$(\!|\alpha_{|||}|\!) \cdot \mathsf{s} = (\!|\alpha_{|||}|\!)$$

which, by fusion, is implied by

$$\alpha_{|||} \cdot \mathsf{s} = \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|||}$$

Proofs and Calculation

$$
\begin{aligned}
& \alpha_{|||} \cdot \mathsf{s} \\
&= \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \triangle \cdot \mathsf{s} \\
&= \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot (\mathsf{s} \times \mathsf{s}) \cdot \triangle \\
&= \cup \cdot (\tau_r \cdot \mathsf{s} \times \tau_l \cdot \mathsf{s}) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle \\
&= \cup \cdot (\mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \tau_l \times \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle \\
&= \cup \cdot (\mathcal{P}(\mathsf{id} \times \mathsf{s}) \times \mathcal{P}(\mathsf{id} \times \mathsf{s})) \cdot (\tau_l \times \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle \\
&= \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot \mathsf{s} \cdot (\tau_l \times \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle \\
&= \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \mathsf{s} \cdot \triangle \\
&= \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \triangle \\
&= \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|||}
\end{aligned}
$$

Process Calculi

Synchronous Product

- models the simultaneous execution of its two arguments
- at each step the resulting action is determined by the interaction structure for the calculus $||| = [\![\alpha_{|||}]\!]$

$$\alpha_{\otimes} = \nu \times \nu \xrightarrow{(\omega \times \omega)} \mathcal{P}(Act \times \nu) \times \mathcal{P}(Act \times \nu)$$

$$\xrightarrow{\delta_r} \mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\mathsf{sel}} \mathcal{P}(Act \times (\nu \times \nu))$$

where $\mathsf{sel} = \mathsf{filter}_{\{0\}}$ filters out all synchronisation failures

Process Calculi

- interaction is catered by δ_r — the distributive law for the strong monad $\mathcal{P}(Act \times \mathsf{Id})$

$$\delta_r^{\mathcal{P}(Act \times \mathsf{Id})} \langle c_1, c_2 \rangle = \{ \langle a' \theta a, \langle p, p' \rangle \rangle | \; \langle a, p \rangle \in c_1 \; \wedge \; \langle a', p' \rangle \in c_2 \}$$

- the underlying Kleisli composition involves the application of the monad multiplication to 'flatten' the result which, for a monoid monad, requires the suitable application of the underlying monoidal operation: here this fixes the interaction discipline

Process Calculi

$$\delta_r^{\mathcal{P}(Act\times\mathsf{Id})}$$

$$= \quad \{ \ \delta_r \text{ definition } \}$$

$$\tau_r^{\mathcal{P}(Act\times\mathsf{Id})} \bullet \tau_l^{\mathcal{P}(Act\times\mathsf{Id})}$$

$$= \quad \{ \ \bullet \text{ definition } \}$$

$$\mu^{\mathcal{P}(Act\times\mathsf{Id})} \cdot \mathcal{P}(\mathsf{id} \times \tau_r^{\mathcal{P}(Act\times\mathsf{Id})}) \cdot \tau_l^{\mathcal{P}(Act\times\mathsf{Id})}$$

$$= \quad \{ \ \mu \text{ for } \mathcal{P}(Act \times \mathsf{Id}) \ \}$$

$$\eta^{\mathsf{B}_1\ \mathsf{B}_2} = \eta^{\mathsf{B}_1} \cdot \eta^{\mathsf{B}_2}$$

$$\mu^{\mathsf{B}_1\ \mathsf{B}_2} = \mathsf{B}_1\ \mu^{\mathsf{B}_2} \cdot \mu^{\mathsf{B}_1} \cdot \mathsf{B}_1\ \gamma$$

Process Calculi

with $\gamma : \mathsf{B}_2\ \mathsf{B}_1 \longrightarrow \mathsf{B}_1\ \mathsf{B}_2$
in this case $\gamma = \tau_l^{\mathsf{B}} : Act \times \mathcal{P}\mathsf{Id} \longrightarrow \mathcal{P}(Act \times \mathsf{Id})$
Then,

$$\mathcal{P}\mu^{(Act\times\mathsf{Id})} \cdot \mu^{\mathcal{P}} \cdot \mathcal{P}\tau_l^{\mathcal{P}} \cdot \mathcal{P}(\mathsf{id} \times \tau_r^{\mathcal{P}(Act\times\mathsf{Id})}) \cdot \tau_l^{\mathcal{P}(Act\times\mathsf{Id})}$$

$$= \quad \{ \ \mu \text{ for } (Act \times \mathsf{Id}) \ \}$$

$$\mathcal{P}((\theta \times \mathsf{id}) \cdot \mathsf{a}^\circ) \cdot \mu^{\mathcal{P}} \cdot \mathcal{P}\tau_l^{\mathcal{P}} \cdot \mathcal{P}(\mathsf{id} \times \tau_r^{\mathcal{P}(Act\times\mathsf{Id})}) \cdot \tau_l^{\mathcal{P}(Act\times\mathsf{Id})}$$

$$= \quad \{ \ \mu^{\mathcal{P}} \text{ natural and definition } \}$$

$$\bigcup \cdot \mathcal{P}(\mathcal{P}((\theta \times \mathsf{id}) \cdot \mathsf{a}^\circ) \cdot \tau_l^{\mathcal{P}}) \cdot \mathcal{P}(\mathsf{id} \times \tau_r^{\mathcal{P}(Act\times\mathsf{Id})}) \cdot \tau_l^{\mathcal{P}(Act\times\mathsf{Id})}$$

Process Calculi

$$\mathcal{P}(Act \times \nu) \times \mathcal{P}(Act \times \nu) \xrightarrow{\tau_l^{\mathcal{P}}} \mathcal{P}(\mathcal{P}(Act \times \nu) \times (Act \times \nu))$$

$$\xrightarrow{\tau_l^{Act \times \mathsf{Id}}} \mathcal{P}(Act \times (\mathcal{P}(Act \times \nu) \times \nu))$$

$$\xrightarrow{\mathcal{P}(\mathsf{id} \times \tau_r^{\mathcal{P}})} \mathcal{P}(Act \times \mathcal{P}((Act \times \nu) \times \nu))$$

$$\xrightarrow{\mathcal{P}(\mathsf{id} \times \mathcal{P}\tau_r^{Act \times \mathsf{Id}})} \mathcal{P}(Act \times \mathcal{P}(Act \times (\nu \times \nu)))$$

$$\xrightarrow{\tau_l^{\mathcal{P}}} \mathcal{P}\mathcal{P}(Act \times (Act \times (\nu \times \nu)))$$

Process Calculi

$$\xrightarrow{\mathcal{P}\mathcal{P}\mathsf{a}^\circ} \mathcal{P}\mathcal{P}((Act \times Act) \times (\nu \times \nu))$$

$$\xrightarrow{\mathcal{P}\mathcal{P}(\theta \times \mathsf{id})} \mathcal{P}\mathcal{P}(Act \times (\nu \times \nu))$$

$$\xrightarrow{\bigcup} \mathcal{P}(Act \times (\nu \times \nu))$$

Process Calculi

Encoding

```
def bsyn{eqL: Lb(N) * Lb(N) -> bool}:
        Pr(Ac(Lb(N))) * Pr(Ac(Lb(N)))
        -> Pr(Ac(Lb(N)))

 = (t1, t2) =>
      (| (r1,r2) => bh:
         sel{eqL} deltar{eqL} (bh r1, bh r2)
      |) (t1,t2).
```


Process Calculi

Parallel Composition

- combines interleaving with synchronous product
- ... to cater for both the individual derivation plus the synchronisations allowed by θ
- ... but performed at the genes level

$$
\begin{aligned}
\alpha_{|} \;=\; \nu \times \nu &\xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) \\
&\xrightarrow{(\alpha_{|||} \times \alpha_{\otimes})} \mathcal{P}(Act \times (\nu \times \nu)) \times \mathcal{P}(Act \times (\nu \times \nu)) \\
&\xrightarrow{\cup} \mathcal{P}(Act \times (\nu \times \nu))
\end{aligned}
$$

Process Calculi

Encoding

```
def bpar{eqL: Lb(N) * Lb(N) -> bool}:
        Pr(Ac(Lb(N))) * Pr(Ac(Lb(N))) -> Pr(Ac(Lb(N)))

 = (t1, t2) =>
   (| (r1,r2) => bh: union(
                 sel{eqL} deltar{eqL} (bh r1, bh r2),
                 union(taur(bh r1, r2), taul(bh r2, r1)))
    |) (t1,t2).
```


Process Calculi

Laws ...

But now focus on proofs!

Proofs and Calculation

- Replace the explicit construction of bisimulations by equational reasoning
- essentially pointfree ...
- ... used in functional programming (*cf.*, Bird-Meertens school)

dynamic combinators	$\rightsquigarrow$	simple equational proofs
static combinators	$\rightsquigarrow$	proofs by fusion
conditional laws	*require*	a form of conditional fusion

Proofs and Calculation

$\boxed{\text{How can conditional laws be derived?}}$

For example, the following classic law in Ccs:

$$(p \mid q)\backslash_K = p \backslash_K | q \backslash_K \qquad \Leftarrow \mathcal{L}(p) \cap \overline{\mathcal{L}(p)} \cap (K \cup \overline{K}) = \emptyset$$

or the simple fact that renaming absent actions has no effect

Proofs and Calculation

In several cases, but not in all, renaming with $f = \{b/a\}$ has no effect.

$$
\begin{array}{ll}
 & [f] = \mathsf{id} \\
\equiv & \quad \{ \text{ renaming definition and ana reflection } \} \\
 & [\!(\alpha_{[f]})\!] = [\!(\omega)\!] \\
\Leftarrow & \quad \{ \text{ ana fusion and identity } \} \\
 & \alpha_{[f]} = \omega
\end{array}
$$

Intuitively, the last equality holds if a does not show up as an action in the immediate continuations of the process:

Proofs and Calculation

$$\phi \quad = \quad =_{\emptyset} \cdot \cap \cdot \langle \mathcal{P}\pi_1 \cdot \omega, \mathsf{sing} \cdot \underline{a} \rangle$$

But ϕ is stated as a local condition

To be taken as a sufficient condition for $[f] = \mathsf{id}$, it has to be lifted to β-invariant.

Proofs and Calculation

$\boxed{\text{The conditional fusion theorem [Bar01]}}$

Let α and β be T-coalgebras and ϕ a predicate on the carrier of β. Then

$$(\phi \Rightarrow (\alpha \cdot h = \mathsf{T}\ h \cdot \beta)) \;\Rightarrow\; (\Box_\beta\ \phi \Rightarrow ([\![(\alpha)]\!]_{\mathsf{T}} \cdot h = [\![(\beta)]\!]_{\mathsf{T}}))$$

where $\Box_\beta$ is the *greatest* fix point of $\lambda_x .\ \phi \cap \bigcirc_\beta\ x$ [Jacobs99]
Informally, $\Box_\beta\ \phi$ reads:

ϕ holds now and in all successor states under β dynamics

Proofs and Calculation

A predicate $\phi : U \longrightarrow \mathbf{2}$ over the carrier of a T-coalgebra $\langle U, p : U \longrightarrow \mathsf{T}\ U\rangle$ is a p-invariant if it is closed under the coalgebra dynamics:

$$\phi \;\Rightarrow\; \bigcirc_p\ \phi$$

where $\bigcirc_p$:

$$(\bigcirc_p\ \phi)\ u \quad \text{iff} \quad \forall_{u' \in_{\mathsf{T}} p\ u} .\ \phi\ u'$$

Informally:

$\bigcirc_p\ \phi$ holds in all states whose immediate successors under p satisfy ϕ

Proofs and Calculation

Or, in terms of predicate lifting (cf, [Jacobs99])

$$\bigcirc_p \phi = \{u \in U |\ p\ u \in (\phi)^{\mathsf{T}}\}$$

T	$(\phi)^{\mathsf{T}}$
Id	ϕ
$\underline{K}$	K
$\mathsf{T}_1 \times \mathsf{T}_2$	$\{\langle x_1, x_2\rangle |\ x_1 \in (\phi)^{\mathsf{T}_1} \wedge x_2 \in (\phi)^{\mathsf{T}_2}\}$
$\mathsf{T}_1 + \mathsf{T}_2$	$\{\iota_1\ x_1 |\ x_1 \in (\phi)^{\mathsf{T}_1}\} \cup \{\iota_2\ x_2 |\ x_2 \in (\phi)^{\mathsf{T}_2}\}$
T^K	$\{f |\ \forall_{k \in K} .\ f\ k \in (\phi)^{\mathsf{T}}\}$
$\mathcal{P}\mathsf{T}$	$\{t |\ \forall_x\ .\ x \in t \Rightarrow x \in (\phi)^{\mathsf{T}}\}$

Proofs and Calculation

Proof

Let X be the carrier of β and i_ϕ the embedding of the subset of X classified by ϕ, *i.e.*, $\phi \cdot i_\phi = \underline{true} \cdot !$. Recall that any β-invariant induces a subcoalgebra β': therefore, $i_{\Box_\beta\ \phi}$ becomes a comorphism from β' to β. Then

$$
\begin{array}{ll}
 & \phi \Rightarrow (\alpha \cdot h = \mathsf{T}\ h \cdot \beta) \\
\equiv & \qquad \{\ i_\phi \text{ definition }\} \\
 & \alpha \cdot h \cdot i_\phi = \mathsf{T}\ h \cdot \beta \cdot i_\phi \\
\Rightarrow & \qquad \{\ \Box_\beta\ \phi \subseteq \phi\ \} \\
 & \alpha \cdot h \cdot i_{\Box_\beta\ \phi} = \mathsf{T}\ h \cdot \beta \cdot i_{\Box_\beta\ \phi} \\
\equiv & \qquad \{\ i_{\Box_\beta\ \phi} \text{ is a comorphism from } \beta' \text{ to } \beta\}
\end{array}
$$

Proofs and Calculation

$$\alpha \cdot h \cdot i_{\Box_\beta\ \phi} \;=\; \mathsf{T}\ h \cdot \mathsf{T}\ i_{\Box_\beta\ \phi} \cdot \beta'$$

$\equiv$ { functoriality}

$$\alpha \cdot h \cdot i_{\Box_\beta\ \phi} \;=\; \mathsf{T}\ (h \cdot i_{\Box_\beta\ \phi}) \cdot \beta'$$

$\equiv$ { fusion}

$$[\![\alpha]\!]_{\mathsf{T}} \cdot h \cdot i_{\Box_\beta\ \phi} \;=\; [\![\beta']\!]_{\mathsf{T}}$$

$\equiv$ { $i_{\Box_\beta\ \phi}$ being a comorphism implies $[\![\beta']\!]_{\mathsf{T}} = [\![\beta]\!]_{\mathsf{T}} \cdot i_{\Box_\beta\ \phi}$}

$$[\![\alpha]\!]_{\mathsf{T}} \cdot h \cdot i_{\Box_\beta\ \phi} \;=\; [\![\beta]\!]_{\mathsf{T}} \cdot i_{\Box_\beta\ \phi}$$

$\equiv$ { $i_{\Box_\beta\ \phi}$ definition }

$$\Box_\beta\ \phi \Rightarrow ([\![\alpha]\!]_{\mathsf{T}} \cdot h = [\![\beta]\!]_{\mathsf{T}})$$

Proofs and Calculation

Unfolding $\Box_\beta\ \phi$

$$[\{b/a\}] \;=\; \mathsf{id} \;\Leftarrow\; \Box_\omega\ \phi$$

for $\Box_\omega\ \phi$ the greatest fixpoint of

$$\Phi \;=\; \lambda x\,.\ \phi \cap \bigcirc_\omega\ x$$

I.e., by Tarski theorem, the union of all post-fixpoints of Φ:

$$\Box_\omega\ \phi \;=\; \bigcup\{s \in \mathcal{P}\nu |\ s \subseteq \phi \cap \bigcirc_\omega\ s\}$$

Proofs and Calculation

Then,

$$
\begin{aligned}
p \in s \;\Rightarrow\; & p \in \phi \wedge p \in \bigcirc_\omega\; s \\
\equiv\;& p \in \phi \wedge p \in \{x \in \nu |\; \omega\; x \in (s)^{\mathcal{P}(Act \times \_)}\} \\
\equiv\;& p \in \phi \wedge p \in \{x \in \nu |\; \omega\; x \in \{c \in \mathcal{P}(Act \times \nu) |\; \forall_{t \in Act \times \nu}\,.\, t \in c \Rightarrow t \in (s)^{Act \times \_}\}\} \\
\equiv\;& p \in \phi \wedge p \in \{x \in \nu |\; \omega\; x \in \{c \in \mathcal{P}(Act \times \nu) |\; \forall_{t \in Act \times \nu}\,.\, t \in c \Rightarrow \pi_2\; t \in s\}\} \\
\equiv\;& p \in \phi \wedge p \in \{x \in \nu |\; (\mathcal{P}\pi_2 \cdot \omega)\; x \in s\}
\end{aligned}
$$

Seen as a set, predicate ϕ is given by
$\phi = \{x \in \nu |\; (\mathcal{P}\pi_1 \cdot \omega)\; x \cap \{a\} = \emptyset\}$.

Proofs and Calculation

Therefore,

$$\square_\omega\; \phi \;=\; \bigcup\{s \in \mathcal{P}\nu |\; \forall_{x \in \nu}\,.\, x \in s \Rightarrow ((\mathcal{P}\pi_1 \cdot \omega)\; x \cap \{a\} = \emptyset) \wedge ((\mathcal{P}\pi_2 \cdot \omega)\; x \in s))\}$$

In words: the set of all processes whose derivations never exhibit action a.

Proofs and Calculation

In CCS,

$$[\{b/a\}] = \mathsf{id} \Leftarrow \notin_a \cdot \mathcal{L}$$

where $\notin_a = \lambda x \,.\, a \notin x$

Going pointwise $p\,[\{b/a\}] = p \quad \Leftarrow a, \overline{a} \notin \mathcal{L}(p)$
where $\mathcal{L}(p)$ is the sort of p defined sematically:

$$\mathcal{L}(p) = (\mathsf{filter}_{Act-L} \cdot \mathcal{P}\pi_1 \cdot \bigcup \cdot \mathcal{P}\omega)\ \ ((\Box_\omega\ \mathsf{tt})\ p)$$

Proofs and Calculation

A Derivation Example

$$\backslash_K \cdot \otimes = \otimes \cdot (\backslash_K \times \backslash_K) \qquad \Leftarrow \mathsf{uniform\_restriction}$$

The proof relies on an immediate consequence of fusion:

to prove the equality $\phi = \psi$ it is enough to show that both $\omega \cdot \phi = \mathsf{T}\ \phi \cdot \alpha$

and $\omega \cdot \psi = \mathsf{T}\ \psi \cdot \alpha$ hold

Proofs and Calculation

Unfolding RHS

$$
\begin{aligned}
& \omega \cdot \otimes \cdot (\backslash_K \times \backslash_K) \\
= & \quad \{ \text{ comorphism, definition} \} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \cdot (\backslash_K \times \backslash_K) \\
= & \quad \{ \text{ functoriality} \} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \cdot \backslash_K \times \omega \cdot \backslash_K) \\
= & \quad \{ \text{ comorphism, functoriality} \} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathcal{P}(\mathsf{id} \times \backslash_K) \times \mathcal{P}(\mathsf{id} \times \backslash_K)) \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}) \\
= & \quad \{ \ \delta_r \text{ naturality} \}
\end{aligned}
$$

Proofs and Calculation

Unfolding RHS

$$
\begin{aligned}
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mathcal{P}(\mathsf{id} \times (\backslash_K \times \backslash_K)) \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}) \\
= & \quad \{ \ \mathsf{sel} \text{ definition, functoriality} \} \\
& \mathcal{P}(\mathsf{id} \times \otimes \cdot (\backslash_K \times \backslash_K)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})
\end{aligned}
$$

Proofs and Calculation

Unfolding LHS

$$
\begin{array}{ll}
& \omega \cdot \backslash_K \cdot \otimes \\
= & \qquad \{\text{ comorphism, definition}\} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \omega \cdot \otimes \\
= & \qquad \{\text{ comorphism}\} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_\otimes \\
= & \qquad \{\text{ definition}\} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \\
= & \qquad \{\text{ } \mathsf{filter}_K \text{ naturality}\}
\end{array}
$$

Proofs and Calculation

Unfolding LHS

$$
\begin{array}{ll}
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{filter}_K \cdot \mathsf{sel} \cdot \delta_r \cdot (\omega \times \omega) \\
\stackrel{?}{=} & \qquad \{\ \star\ \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathsf{filter}_K \times \mathsf{filter}_K) \cdot (\omega \times \omega)
\end{array}
$$

Proofs and Calculation

Last step requires:

$$\mathsf{filter}_K \cdot \mathsf{sel} \cdot \delta_r \;=\; \mathsf{sel} \cdot \delta_r \cdot (\mathsf{filter}_K \times \mathsf{filter}_K) \qquad (*)$$

Then,

$$\begin{aligned}
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathsf{filter}_K \times \mathsf{filter}_K) \cdot (\omega \times \omega) \\
= & \quad \{ \text{ functoriality } \} \\
& \mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\mathsf{filter}_K \cdot \omega \times \mathsf{filter}_K \cdot \omega) \\
= & \quad \{ \text{ definition} \} \\
& \mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \otimes)) \cdot \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})
\end{aligned}$$

and conclude by conditional fusion, generating the suitable invariant

Proofs and Calculation

Unfolding (*) to find the invariant

$$\begin{aligned}
& (\,\mathsf{filter}_K \cdot \mathsf{sel} \cdot \delta_r)\,\langle c_1, c_2\rangle \;= \\
& = (\mathsf{filter}_K \cdot \mathsf{sel})\,\{\langle a'\theta a, \langle p, p'\rangle\rangle |\ \langle a,p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2\} \\
& = \mathsf{filter}_K\,\{\langle a'\theta a, \langle p, p'\rangle\rangle |\ \langle a,p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2 \;\wedge\; a'\theta a \neq 0\} \\
& = \{\langle a'\theta a, \langle p, p'\rangle\rangle |\ \langle a,p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2 \;\wedge\; a'\theta a \neq 0 \;\wedge\; a'\theta a \notin K\}
\end{aligned}$$

and

$$\begin{aligned}
& (\,\mathsf{sel} \cdot \delta_r \cdot (\mathsf{filter}_K \times \mathsf{filter}_K))\,\langle c_1, c_2\rangle \;= \\
& = (\mathsf{sel} \cdot \delta_r)\;\langle\{\langle a,p\rangle \in c_1 |\ a \notin K\}, \{\langle a', p'\rangle \in c_2 |\ a' \notin K\}\rangle \\
& = \mathsf{sel}\;\{\langle a'\theta a, \langle p, p'\rangle\rangle |\ \langle a,p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2 \;\wedge\; a, a' \notin K\} \\
& = \{\langle a'\theta a, \langle p, p'\rangle\rangle |\ \langle a,p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2 \;\wedge\; a, a' \notin K \;\wedge\; a'\theta a \neq 0\}
\end{aligned}$$

Proofs and Calculation

Step $\star$ requires restriction to pairs of processes such that:

$$\phi\ \langle p, q \rangle \ = $$
$$\forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\ p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ a\theta a' \neq 0 \ \Rightarrow\ (a\theta a' \notin K \equiv a, a' \notin K)$$

which is lifted to the invariant

$$\mathsf{uniform\_restriction} \ = \ \Box_\alpha\ \phi$$

where $\alpha = \mathsf{sel} \cdot \delta_r \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})$

Finally ϕ must be interpreted wrt the interaction structure

The Ccs Case

$$\forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\ p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ a\theta a' \neq 0 \ \Rightarrow\ (a\theta a' \notin K \ \equiv\ a, a' \notin K)$$

$\equiv$ { Ccs interaction structure }

$$\forall_{a \in \mathcal{P}(\pi_1 \cdot \omega)\ p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ (a\theta a' = \tau \ \vee\ a\theta a' = 1) \ \Rightarrow\ a, a' \notin K$$

$\equiv$ { $a\theta a' = 1$ iff $a = a' = 1$ }

$$\forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\ p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ a\theta a' = \tau \ \Rightarrow\ a, a' \notin K$$

$\equiv$ { $a\theta a' = \tau$ iff $a' = \overline{a}$}

$$\forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\ p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\ q}\ .\ a' = \overline{a} \ \Rightarrow\ a, a' \notin K$$

$\equiv$ { rearranging}

$$(\mathcal{P}\pi_1 \cdot \omega)\ p \cap \overline{(\mathcal{P}\pi_1 \cdot \omega)\ q} \cap (K \cup \overline{K}) = \emptyset$$

Proofs and Calculation

The CSP case

In this case $a\theta a' \neq 0 \Rightarrow (a\theta a' \notin K \;\equiv\; a, a' \notin K)$ is trivially true and the law holds without any side condition

Remarks

- Constraints are discovered rather than postulated
- They depend essentially on the chosen interaction structure

Proofs and Calculation

Proof Re-Use: $\backslash_K \cdot | \;=\; | \cdot (\backslash_K \times \backslash_K)$

A simple calculation leads to

$$\backslash_K \cdot | \;=\; | \cdot (\backslash_K \times \backslash_K) \qquad \Leftarrow \; \mathsf{uniform\_restriction}'$$

Note that in $\mathsf{uniform\_restriction}'$ ϕ is closed wrt the transitions on α', which includes both interleavings and synchronisations leading to

$$(p \mid q)\backslash_K = p\,\backslash_K \,| q\,\backslash_K \qquad \Leftarrow \mathcal{L}(p) \cap \overline{\mathcal{L}(p)} \cap (K \cup \overline{K}) = \emptyset$$

A Process Language

- ... to deal with recursive processes
- requires process variables and an environment
- modelled by a collection of associations $v = exp$ where exp is a guarded process expression to ensure existence of unique solutions
- as well as a prefix-like construction — $\mathtt{pvar}$ — to introduce (guarded) variables in an expression

A Process Language

- introduce $\mathtt{Ln\,(L)}$ as the initial algebra for a functor Σ underlying

$$
\begin{array}{rclclcl}
\langle \mathtt{P} \rangle & ::= & \mathtt{pnil} & | & \mathtt{ppre}\,(a, \langle \mathtt{P} \rangle) & | & \mathtt{pcho}\,(\langle \mathtt{P} \rangle, \langle \mathtt{P} \rangle) \quad | \\
 & & \mathtt{pint}\,(\langle \mathtt{P} \rangle, \langle \mathtt{P} \rangle) & | & \mathtt{psyn}\,(\langle \mathtt{P} \rangle, \langle \mathtt{P} \rangle) & | & \mathtt{ppar}\,(\langle \mathtt{P} \rangle, \langle \mathtt{P} \rangle) \quad | \\
 & & \mathtt{pret}\,(\langle \mathtt{P} \rangle, K) & | & \mathtt{pren}\,(\langle \mathtt{P} \rangle, f) & | & \mathtt{pvar}\,(a, i)
\end{array}
$$

where

$a \in \mathtt{Ac\,(L)}$

$K \subseteq \mathtt{L}$

i is a process variable and f a renaming $\mathtt{Ac\,(L)}$-homomorphism

A Process Language

The definition is generic:

- parametric on `L`
- resorts to whatever interaction structure is provided for `Ac(L)`

```
data Ln(L) -> P =
      pnil: 1 -> P | ppre: Ac(L) * P -> P |
      pcho: P * P -> P | pint: P * P -> P |
      psyn: P * P -> P | ppar: P * P -> P |
      pret: P * set(L) -> P                |
      pren: P * map(Ac(L), Ac(L)) -> P     |
      pvar: Ac(L) * string -> P.
```


The Interpreter

How can `Ln(L)` expressions be interpreted as processes?

- initial algebra approach: once the syntax is fixed, a semantic Σ-algebra would be produced and the interpretation defined as the associated catamorphism
- final coalgebra approach:
  - start from a *semantic universe*: the final coalgebra for $\mathcal{P}(\mathtt{Ac(L)} \times \mathsf{Id})$
  - cast the syntax — the set of terms `Ln(L)` — into a $\mathcal{P}(\mathtt{Ac(L)} \times \mathsf{Id})$-coalgebra which computes the 'syntactical' derivations of process terms
  - the interpreter is the associated *anamorphism*

The Interpreter

$$
\begin{tikzcd}
\mathtt{Pr(Ac(L))} \arrow[r, "\omega"] & \mathcal{P}(\mathtt{Ac(L)} \times \mathtt{Pr(Ac(L))}) \\
\mathtt{Ln(L)} \arrow[u, "\mathtt{sem}"] \arrow[r, "\mathtt{syn}"] & \mathcal{P}(\mathtt{Ac(L)} \times \mathtt{Ln(L)}) \arrow[u, "\mathcal{P}(\mathsf{id} \times \mathtt{sem})"'] \\
\Sigma\ \mathtt{Ln(L)} \arrow[u, "\alpha"] \arrow[r, "\Sigma\ \mathtt{syn}"] & \Sigma\ \mathcal{P}(\mathtt{Ac(L)} \times \mathtt{Ln(L)}) \arrow[u, "\alpha_{\mathtt{syn}}"']
\end{tikzcd}
$$

$$\mathtt{sem} = [\!(\mathtt{syn})\!]_{\mathcal{P}(\mathtt{Ac(L)} \times \mathsf{Id})}$$

comes for free given coalgebra $\mathtt{syn}$ which is defined in a ‘canonical’ way as catamorphism

$$\mathtt{syn} = (\!|\alpha_{\mathtt{syn}}|\!)_{\Sigma}$$

Environment

- modeled as $E = \mathtt{map(string,\ Ln(L))}$
- the semantic function becomes typed as:

$$\mathtt{sem} : \mathtt{Ln(L)} \times E \longrightarrow \mathtt{Pr(Ac(L))}$$

- I.e.,
  - $\mathtt{sem}$ — a strong anamorphism for $\mathcal{P}(\mathtt{Ac(L)} \times \mathsf{Id})$
  - $\mathtt{syn}$ — a strong catamorphism for Σ
- the diagram above remains valid, but it has to be interpreted in the Kleisli category for the product comonad
whose final coalgebra is $\omega' = \omega \cdot \pi_1$ and initial algebra $\alpha' = \alpha \cdot \pi_1$

Environment

$$
\begin{tikzcd}
\mathtt{Pr\,(Ac\,(L))} \times E \arrow[rrr, "\omega'"] \arrow[d, dash, dotted] & & & \mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Pr\,(Ac\,(L))}) \\
\mathtt{Pr\,(Ac\,(L))} & & & \\
\mathtt{Ln\,(L)} \times E \arrow[u, "\mathtt{sem}"] \arrow[rr, "\mathtt{syn}"] \arrow[d, dash, dotted] & & \mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Ln\,(L)}) \arrow[r, dash, dotted] & \mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Ln\,(L)}) \times E \arrow[uu] \\
\mathtt{Ln\,(L)} & & & \\
\Sigma\,\mathtt{Ln\,(L)} \times E \arrow[u, "\alpha'"] \arrow[r] & \Sigma\,\mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Ln\,(L)}) \arrow[r, dash, dotted] & \Sigma\,\mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Ln\,(L)}) \times E \arrow[uu, "\alpha_{\mathtt{syn}}"'] &
\end{tikzcd}
$$

Environment

Summing up

$$\mathtt{sem} \;=\; \mathsf{unfold}_{\mathcal{P}(\mathtt{Ac\,(L)} \times \mathsf{Id})}\; \mathtt{syn}$$

where

$$\mathtt{syn} \;=\; \mathsf{fold}_{\Sigma}\, \alpha_{\mathtt{syn}}$$

- However, $\mathtt{syn}$ must be defined in terms of both the computations on the substructures of its argument and these substructures themselves
- i.e., as a strong paramorphism

Environment

In general, a paramorphisms is the unique arrow st

$$\begin{tikzcd}
X & \mathsf{T}\,(X \times \mu_{\mathsf{T}}) \arrow[l, "f"'] \\
\mu_{\mathsf{T}} \arrow[u, "\mathsf{par}_{\mathsf{T}}\ f"] & \mathsf{T}\,\mu_{\mathsf{T}} \arrow[l, "\alpha_{\mathsf{T}}"] \arrow[u, "\mathsf{T}\,\langle \mathsf{par}_{\mathsf{T}}\ f, \mathsf{id}\rangle"']
\end{tikzcd}$$

Adding strength results in

$$\alpha_{\mathsf{syn}} : \Sigma\,((\mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Ln\,(L)}))\times\mathtt{Ln\,(L)}) \times E \longrightarrow \mathcal{P}(\mathtt{Ac\,(L)} \times \mathtt{Ln\,(L)})$$

dealt in CHARITY through the # annotation: inside a fold it stands for the value being currently analysed, before the recursive application

Environment

```
def sem{eqL: Lb(N) * Lb(N) -> bool}:
     Ln(Lb(N)) * map(string, Ln(Lb(N)))  -> Pr(Ac(Lb(N)))

    = (exp,m)  => (| e => bh: syn{eqL}(e,m) |) exp.
```

and

```
def syn{eqL: Lb(N) * Lb(N) -> bool}:
     Ln(Lb(N)) * map(string, Ln(Lb(N)))
     ->  set(Ac(Lb(N)) * Ln(Lb(N)))
```

Environment

```
= (pr,m) =>
  {| pnil: ()      => empty
   | ppre: (a,l)   => sing(a, p1 #)
   | pvar: (a,s)   => { ss(p) => sing(a, p)
                      | ff    => empty
                      } app{eq_string}(m,s)
   | pcho: (lp,lq) => union(lp, lq)
   | pint: (lp,lq) => union(stau1(lp, p1 #), stau1(lq,  p0 #))
   | psyn: (lp,lq) => ssel{eqL} sdelta1{eqL} (lp,lq)
   | ppar: (lp,lq) => union(union(stau2(lp, p1 #), stau2(lq,  p0 #)),
                            ssel{eqL} sdelta2{eqL} (lp,lq))
   | pret: (l,k)   => set{x => (p0 x, pret(p1 x, k))}
                filter{x => not member{eqA{eqL}}(p0 x, compren k)} l
   | pren: (l,h)   => set{x => { ff    => (p0 x,  pren(p1 x, h))
                               | ss a  => (a, pren(p1 x, h))
                               } app{eqA{eqL}}(compret h, p0 x) } l
   |} pr.
```


Conclusions

- An exercise in the pragmatics of resorting to a formal duality — that of algebra and coalgebra — to explain the intuitive symmetry between data and behavioural structures
- leads to a simple strategy to functional prototyping of process calculi by direct implementation of their semantics in CHARITY
- with didactic applications
- ... and genericity as the *driving motto*

Conclusions

at different levels:

- *behaviour model*, e.g.

```
data C -> PartialPr(A) = bh: C -> SF(A * C).

data C -> OrderedPr(A) = bh: C -> list(A * C).
```

- *interaction discipline*
- transition *structure*, e.g.

```
data C -> ReactivePr(A) = bh: C -> A => set(C).
```


Current Work

- Replace the explicit construction of bisimulations by equational, basically pointfree reasoning [CMCS'01]
- Systematic study of co-recursion schemes and suitable semantic categories
- Open questions: naming and mobility
- Development of a prototyping kernel with a graphical interface based on the coinductive approach [on-going Ribeiro's MSc thesis]