Probabilistic Modular Embedding for Stochastic Coordinated Systems

Stefano Mariani, Andrea Omicini
{s.mariani, andrea.omicini}@unibo.it

Dipartimento di Informatica: Scienza e Ingegneria (DISI)
Alma Mater Studiorum—Università di Bologna

COORDINATION2013 - Firenze, Italy, 3-5/6/2013

1. Motivations & Background
2. Probabilistic Modular Embedding
   - Probabilistic Observation
   - Probabilistic Termination
   - Formal Tools
3. Early Case Studies
   - `ProbLinCa` vs. `LinCa`
   - pKLAIM vs. KLAIM
   - π_{pa}-*calculus* vs. π_a-*calculus*
4. Conclusion & Further Works

Outline

Comparing Languages Expressiveness

- Understanding *expressiveness* of coordination languages is essential to deal with *interactions* complexity [Wegner, 1997]
- The notion of modular embedding [de Boer and Palamidessi, 1994] is particularly effective in comparing the *relative expressiveness* of concurrent languages

A new kind of systems

However, the emergence of systems featuring stochastic behaviours [Omicini and Viroli, 2011] is asking for new techniques to observe, model and measure their expressiveness.

Shapiro's Embedding

The informal definition of *embedding* assumes that a language could be translated in another [Shapiro, 1991]:

- **Easily** — "without the need for a global reorganisation of the program"
- **Equivalently** — "without affecting the program's observable behaviour"

Formally

Given two languages L, L', their program sets $Prog_L, Prog_{L'}$, and the powersets of their observable behaviours Obs, Obs', we assume that two observation criteria Ψ, Ψ' hold:

$$\Psi : Prog_L \to Obs \qquad \Psi' : Prog_{L'} \to Obs'$$

Then, *L embeds L'* (written $L \succeq L'$) *iff* there exist a *compiler* $C : Prog_{L'} \to Prog_L$ and a *decoder* $D : Obs \to Obs'$ such that for every program $W \in L'$

$$D(\Psi[C(W)]) = \Psi'[W]$$

Modular Embedding I

- According to [de Boer and Palamidessi, 1994] such definition is too weak, because any pair of Turing-complete languages would embed each other
- So they propose the definition of modular embedding[1] (ME):
  - D can be defined independently for each of all the possible outcomes of all the possible computations:
    $$\forall O \in Obs : D(O) = \{d(o) \mid o \in O\} \text{ (for some } d\text{)}$$
  - In a concurrent setting it is reasonable to require compositionality of the compiler C:
    $$C(A \parallel' B) = C(A) \parallel C(B) \quad , \quad C(A +' B) = C(A) + C(B)$$
    for every pair of programs $A, B \in L'$.
  - Deadlocks should be considered and preserved by decoder D:
    $$\forall O \in Obs, \forall o \in O : tm'(D_{d(o)}) = tm(o)$$
    where tm and tm' are L and L' termination modes.

[1] We stick with symbol $\succeq$ since we assume this definition as our "reference" embedding.

ME Distinguishing Power I

In [Bravetti et al., 2005], the ProbLinCa calculus is defined:

- as the probabilistic extension of the LinCa calculus
- in which each tuple gets a *weight* resembling selection probability: the higher the weight, the higher its matching chance

ProbLinCa vs. LinCa

Suppose the following ProbLinCa process P and LinCa process Q are acting on tuple space S:

$$P = \mathtt{in}_p(T).\emptyset + \mathtt{in}_p(T).\mathtt{rd}_p(T').\emptyset \qquad Q = \mathtt{in}(T).\emptyset + \mathtt{in}(T).\mathtt{rd}(T').\emptyset$$

$$S = \langle \mathtt{t_l}[20], \mathtt{t_r}[10] \rangle$$

where T is a LINDA template matching both tuples $\mathtt{t_l}$ and $\mathtt{t_r}$, whereas T' matches $\mathtt{t_r}$ solely.

ME Distinguishing Power II

From the ME viewpoint, P and Q are *not* distinguishable, being their ending states the same:

$$\Psi[P] = (\texttt{success}, \langle \texttt{t}_\texttt{r}[10] \rangle) \text{ OR } (\texttt{deadlock}, \langle \texttt{t}_\texttt{l}[20] \rangle)$$
$$\Psi[Q] = (\texttt{success}, \langle \texttt{t}_\texttt{r}[10] \rangle) \text{ OR } (\texttt{deadlock}, \langle \texttt{t}_\texttt{l}[20] \rangle)$$

Quantity vs. quality

The point is, that whereas P and Q are qualitatively equivalent, they are not so quantitatively, but ME cannot tell apart the probabilistic information conveyed by, e.g., a `ProbLinCa` primitive w.r.t. a `LinCa` one.

ME Distinguishing Power III

In fact, a "two-way" *modular encoding* can be established by defining compilers C and C' as

$$C_{\texttt{LinCa}} = \begin{cases} \texttt{out} & \longmapsto \texttt{out} \\ \texttt{rd} & \longmapsto \texttt{rd}_p \\ \texttt{in} & \longmapsto \texttt{in}_p \end{cases} \qquad C_{\texttt{ProbLinCa}} = \begin{cases} \texttt{out} & \longmapsto \texttt{out} \\ \texttt{rd}_p & \longmapsto \texttt{rd} \\ \texttt{in}_p & \longmapsto \texttt{in} \end{cases}$$

ME observational equivalence

Therefore,

$$\texttt{ProbLinCa} \succeq \texttt{LinCa} \;\wedge\; \texttt{LinCa} \succeq \texttt{ProbLinCa}$$
$$\Longrightarrow$$
$$\texttt{ProbLinCa} \equiv_{\Psi} \texttt{LinCa}$$

Outline

Redefining "Easily" and "Equivalently" I

Restricting ourselves to *asynchronous coordination calculi*, a process can be said to be *easily* mappable into another if it requires:

1. no extra-computations to mimic complex coordination operators
2. no extra-coordinators (neither coordinated processes nor coordination medium) to handle suspensive semantics
3. no unbounded extra-interactions to perform additional coordination

Focus on coordination primitives

Altogether, such requirements are necessary if the goal is to focus on *"pure coordination expressiveness"*, that is, we intentionally consider coordination primitives solely, abstracting away from processes and coordination media own "algorithmic expressiveness".

Redefining "Easily" and "Equivalently" II

The notions of observables and termination need to be re-casted in the probabilistic setting to re-define the term *equivalently*:

Probabilistic Observation — Observable actions should be associated with their *execution probability*, driven by synchronisation opportunities offered by the coordination medium at run-time.

Probabilistic Termination — Ending states should be defined as those for which all outgoing transitions have probability 0. Furthermore, they should be refined with the probability of reaching that state from a given initial one.

Outline

Probabilistic Observation

Function Θ

Formally, we define the probabilistic observation function (Θ), mapping a process (W) into observables, as follows:

$$\Theta[W] = \left\{ (\rho, W[\bar{\mu}]) \mid \quad (W, \langle\sigma\rangle) \longrightarrow^* (\rho, W[\bar{\mu}]) \right\}$$

where ρ is a probability value $\in [0,1]$, $\bar{\mu}$ is a sequence of actual synchronisations – e.g. $\bar{\mu} = \mu(T_1, t_1), \ldots, \mu(T_n, t_n)$ – and σ is the space state—e.g. $\sigma = \langle t_1, \ldots, t_n \rangle$.

Outline

Probabilistic Termination

Function Φ

Analogously, we define *reachability value* $\rho_\perp$ and the probabilistic termination function Φ as follows:

$$\Phi[W] = \left\{ (\rho_\perp, \tau) \mid \quad (W, \langle\sigma\rangle) \longrightarrow^*_\perp (\rho_\perp, \tau) \right\}$$

where subscript $\perp$ stands for a sequence of finite transitions leading to termination state τ[2].

Notice that Φ abstracts away from computation *traces*, that is, it does not keep track of synchronisations in term $W[\bar{\mu}]$, focussing solely on termination states τ.

[2] E.g. $\tau = \texttt{success}, \texttt{failure}, \texttt{deadlock}, \texttt{undefined}$.

Outline

1. Motivations & Background
2. Probabilistic Modular Embedding
   - Probabilistic Observation
   - Probabilistic Termination
   - Formal Tools
3. Early Case Studies
   - ProbLinCa vs. LinCa
   - pKLAIM vs. KLAIM
   - π_{pa}-*calculus* vs. π_a-*calculus*
4. Conclusion & Further Works

Probability Aggregation Functions

From probability theory:

- the probability of a *sequence* – that is, a *"dot"-separated* list – of probabilistic actions is the *product* of the probabilities of such actions
- the probability of a *choice* – that is, a *"+"-separated* list – of probabilistic actions is the *sum* of the probabilities of such actions

Then we define the sequence probability aggregation function ($\bar{\nu}$) and the choice probability aggregation function (ν^+) as follows:

$\bar{\nu}$ and ν^+ functions

$$\bar{\nu} : W \times \langle\sigma\rangle \mapsto \rho \quad \text{where} \quad \rho = \prod_{j=0}^{n}\{p_j \mid (p_j, \mu_{\bar{\ell}}) \in \Theta[W = \bar{\ell}.W']\}$$

$$\nu^+ : W \times \langle\sigma\rangle \mapsto \rho \quad \text{where} \quad \rho = \sum_{j=0}^{n}\{p_j \mid (p_j, \mu_{\ell^+}) \in \Theta[W = \ell^+.W']\}$$

where $\bar{\ell}$ is a *sequence* of synchronisation actions and ℓ^+ is a *choice* between synchronisation actions.

The "$\uparrow$" Operator I

[Bravetti, 2008] proposes a formalism to deal with those open transition systems which require quantitative information to be attached to synchronisation actions at *run-time*.

The idea is that of *partially closing* labelled transition systems via the process-algebraic operator "$\uparrow$", as follows:

1. actions labelling open transitions are equipped with *handles*
2. $\uparrow$ composes a LTS to a specification G, associating each handle to a given numeric value
3. quantitative information is computed from handle values for each enabled action
4. quantitatively-labelled actions turn an open transition into a *reduction* whose execution is driven by such quantitative information

The "$\uparrow$" Operator II

The $\uparrow$ operator can be used to compute synchronisations probability for Probabilistic Modular Embedding PME, e.g. in the case of `ProbLinCa`:

1. handles represent tuple templates associated to coordination primitives
2. handles listed in term G represent tuples offered by the tuple space (at run-time)
3. G associates handles to their weight
4. closure operator $\uparrow$ matches admissible synchronisations between a process and the tuple space, cutting out unavailable actions and computing admissible ones probability

In Practice I

Given a single probabilistic observable transition step for, e.g., a ProbLinCa process:

$$\mathtt{in}_p(T).P \mid \langle t_1[w_1], .., t_n[w_n] \rangle \quad \xrightarrow{\mu(T,t_j)}_{p_j} \quad P[t_j/T] \mid \langle t_1[w_1], .., t_n[w_n] \rangle \backslash t_j$$

we can expand its reduction steps to unambiguously define its probabilistic semantics:

$$\begin{gathered}\mathtt{in}_p(T).P \mid \langle t_1[w_1], .., t_n[w_n] \rangle \\ \xrightarrow{T} \\ \mathtt{in}_p(T).P \mid \langle t_1[w_1], .., t_n[w_n] \rangle \uparrow \{(t_1, w_1), .., (t_n, w_n)\} \\ \hookrightarrow \\ \mathtt{in}_p(T).P \mid \langle t_1[w_1], .., t_n[w_n] \rangle \uparrow \{(t_1, p_1), .., (t_j, p_j), .., (t_n, p_n)\} \\ \xrightarrow{t_j}_{p_j} \\ P[t_j/T] \mid \langle t_1[w_1], .., t_n[w_n] \rangle \backslash t_j\end{gathered}$$

Coupling this with the probability aggregation functions $\bar{\nu}$ and ν^+, we are now ready to compute PME Θ and Φ functions.

Outline

Outline

ProbLinCa vs. LinCa I

Recall P and Q?

$$P = \mathtt{in}_p(T).\emptyset + \mathtt{in}_p(T).\mathtt{rd}_p(T').\emptyset \qquad Q = \mathtt{in}(T).\emptyset + \mathtt{in}(T).\mathtt{rd}(T').\emptyset$$
$$S = \langle \mathtt{t_l}[20], \mathtt{t_r}[10] \rangle$$

By repeating the embedding observation, but now under the assumptions of PME, we get:

$$\Phi[P] = (0.\bar{6}, \mathtt{success}) \text{ OR } (0.\bar{3}, \mathtt{deadlock})$$
$$\Phi[Q] = (\bullet, \mathtt{success}) \text{ OR } (\bullet, \mathtt{deadlock})$$

where symbol $\bullet$ denotes "absence of information".

ProbLinCa vs. LinCa II

PME tells apart ProbLinCa

This time, only a "one-way" *encoding* can be established, by defining compiler $C_{\texttt{LinCa}}$ as

$$C_{\texttt{LinCa}} = \begin{cases} \texttt{out} & \longmapsto & \texttt{out} \\ \texttt{rd} & \longmapsto & \texttt{rd}_p \\ \texttt{in} & \longmapsto & \texttt{in}_p \end{cases}$$

Therefore, we can state that ProbLinCa *probabilistically embeds* ($\succeq_p$) LinCa, but not the opposite:

$$\texttt{ProbLinCa} \succeq_p \texttt{LinCa} \;\wedge\; \texttt{LinCa} \not\succeq_p \texttt{ProbLinCa}$$
$$\Longrightarrow$$
$$\texttt{ProbLinCa} \not\equiv_p \texttt{LinCa}$$

Outline

pKlaim vs. Klaim I

Klaim [De Nicola et al., 1998] is a kernel programming language for mobile computing.

- processes as well as data can be moved across the network among computing environments
- features Linda with multiple tuple spaces
- *localities* are first-class abstractions to manage mobility and distribution-related aspects

pKlaim

pKlaim [Di Pierro et al., 2004] extends such model through:

- a probabilistic choice operator $+_{i=1}^{n} p_i : P_i$
- a probabilistic parallel operator $|_{i=1}^{n}\ p_i : P_i$
- *probabilistic allocation environments*, formally defined as a partial map $\sigma : Loc \mapsto Dist(S)$ associating probability distributions on physical sites (S) to logical localities (Loc)

pKLAIM vs. KLAIM II

First of all, we focus on the probabilistic choice operator:

$$P = \tfrac{2}{3}\mathtt{in}(T)@s.\emptyset + \tfrac{1}{3}\mathtt{in}(T)@s.\mathtt{rd}(T)@s.\emptyset$$
$$Q = \mathtt{in}(T)@s.\emptyset + \mathtt{in}(T)@s.\mathtt{rd}(T)@s.\emptyset$$
$$s = \mathtt{out(t)@self}.\emptyset \quad \equiv \quad s = \langle \mathtt{t} \rangle$$

Both processes have a non-deterministic branching structure which cannot be distinguished by ME:

$$\Psi[P] = (\mathtt{success}, \langle\,\rangle) \text{ OR } (\mathtt{deadlock}, \langle\,\rangle)$$
$$\Psi[Q] = (\mathtt{success}, \langle\,\rangle) \text{ OR } (\mathtt{deadlock}, \langle\,\rangle)$$

PME tells apart probabilistic choice

PME is instead sensitive to the probabilistic information available for pKLAIM process P:

$$\Phi[P] = (0.\bar{6}, \mathtt{success}) \text{ OR } (0.\bar{3}, \mathtt{deadlock})$$
$$\Phi[Q] = (\bullet, \mathtt{success}) \text{ OR } (\bullet, \mathtt{deadlock})$$

pKLAIM vs. KLAIM III

As regards the probabilistic allocation operator:

$$P = \mathtt{in}(T)@l.\emptyset \qquad Q = \mathtt{in}(T)@l.\emptyset$$

$$s_1 = \langle \mathtt{t} \rangle \quad s_2 = \langle\,\rangle \qquad \sigma : l \mapsto \begin{cases} \frac{2}{3} s_1 \\ \frac{1}{3} s_2 \end{cases}$$

By applying ME we get:

$$\Psi[P] = (\mathtt{success}, s_1 = \langle\,\rangle \land s_2 = \langle\,\rangle) \text{ OR } (\mathtt{deadlock}, s_1 = \langle \mathtt{t} \rangle \land s_2 = \langle\,\rangle)$$
$$\Psi[Q] = (\mathtt{success}, s_1 = \langle\,\rangle \land s_2 = \langle\,\rangle) \text{ OR } (\mathtt{deadlock}, s_1 = \langle \mathtt{t} \rangle \land s_2 = \langle\,\rangle)$$

PME tells apart probabilistic allocation

Whereas ME is *insensitive* to the probabilistic allocation function σ, PME provides "probability-sensitive" observation/termination functions:

$$\Phi[P] = (0.\bar{6}, \mathtt{success}) \text{ OR } (0.\bar{3}, \mathtt{deadlock})$$
$$\Phi[Q] = (\bullet, \mathtt{success}) \text{ OR } (\bullet, \mathtt{deadlock})$$

Outline

π_{pa}-*calculus* vs. π_a-*calculus* I

π_{pa}-*calculus* [Herescu and Palamidessi, 2001] increases the expressive power of π_a-*calculus* [Boudol, 1992] through a probabilistic guarded choice operator ($\sum_i p_i \alpha_i . P_i$) able to distinguish between probabilistic and purely non-deterministic behaviours.

$$P = \left(\tfrac{2}{3}x(y) + \tfrac{1}{3}z(y)\right).\emptyset \qquad Q = \left(x(y) + z(y)\right).\emptyset$$
$$S = \bar{x}y \quad \equiv \quad S = \{S_x = \langle y \rangle \bigcup S_z = \langle\ \rangle\}$$

As expected, they are indistinguishable despite the probabilistic information available for P, which is lost by ME:

$$\Psi[P] = (\texttt{success}, \langle\ \rangle) \text{ OR } (\texttt{deadlock}, \langle y \rangle)$$
$$\Psi[Q] = (\texttt{success}, \langle\ \rangle) \text{ OR } (\texttt{deadlock}, \langle y \rangle)$$

π_{pa}-*calculus* vs. π_a-*calculus* II

PME tells apart π_{pa}-*calculus*

PME fills the gap:

$$\Phi[P] = (0.\bar{6}, \texttt{success}) \text{ OR } (0.\bar{3}, \texttt{deadlock})$$
$$\Phi[Q] = (\bullet, \texttt{success}) \text{ OR } (\bullet, \texttt{deadlock})$$

Outline

Conclusion & Further Works I

- We refined and extended the definition of probabilistic modular embedding (PME) first sketched in [Mariani and Omicini, 2013]
- We showed how PME succeeds in telling apart probabilistic languages from non-probabilistic ones

A first step

Whereas apparently trivial, such a distinction was not possible with any other formal framework in the literature so far, to the best of our knowledge.

Next steps

The ability of PME to tell apart the different probabilistic processes models proposed in [van Glabbeek et al., 1995] is currently under investigation.

Bibliography I

Boudol, G. (1992).
Asynchrony and the Pi-calculus.
Rapport de recherche RR-1702, INRIA.

Bravetti, M. (2008).
Expressing priorities and external probabilities in process algebra via mixed open/closed systems.
Electronic Notes in Theoretical Computer Science, 194(2):31–57.

Bravetti, M., Gorrieri, R., Lucchi, R., and Zavattaro, G. (2005).
Quantitative information in the tuple space coordination model.
Theoretical Computer Science, 346(1):28–57.

de Boer, F. S. and Palamidessi, C. (1994).
Embedding as a tool for language comparison.
Information and Computation, 108(1):128–157.

De Nicola, R., Ferrari, G., and Pugliese, R. (1998).
KLAIM: A kernel language for agent interaction and mobility.
IEEE Transaction on Software Engineering, 24(5):315–330.

Bibliography II

Di Pierro, A., Hankin, C., and Wiklicky, H. (2004).
Probabilistic KLAIM.
In De Nicola, R., Ferrari, G.-L., and Meredith, G., editors, *Coordination Models and Languages*, volume 2949 of *LNCS*, pages 119–134. Springer Berlin / Heidelberg.
6th International Conference (COORDINATION 2004), 24-27 February 2004, Pisa, Italy.

Herescu, O. M. and Palamidessi, C. (2001).
Probabilistic asynchronous pi-calculus.
CoRR, cs.PL/0109002.

Mariani, S. and Omicini, A. (2013).
Probabilistic embedding: Experiments with tuple-based probabilistic languages.
In *28th ACM Symposium on Applied Computing (SAC 2013)*, pages 1380–1382, Coimbra, Portugal.
Poster Paper.

Omicini, A. and Viroli, M. (2011).
Coordination models and languages: From parallel computing to self-organisation.
The Knowledge Engineering Review, 26(1):53–59.
Special Issue 01 (25th Anniversary Issue).

Bibliography III

Shapiro, E. (1991).
Separating concurrent languages with categories of language embeddings.
In *23rd Annual ACM Symposium on Theory of Computing*.

van Glabbeek, R. J., Smolka, S. A., and Steffen, B. (1995).
Reactive, generative, and stratified models of probabilistic processes.
Information and Computation, 121(1):59–80.

Wegner, P. (1997).
Why interaction is more powerful than algorithms.
Communications of the ACM, 40(5):80–91.

Probabilistic Modular Embedding for Stochastic Coordinated Systems

Stefano Mariani, Andrea Omicini
{s.mariani, andrea.omicini}@unibo.it

Dipartimento di Informatica: Scienza e Ingegneria (DISI)
Alma Mater Studiorum—Università di Bologna

COORDINATION2013 - Firenze, Italy, 3-5/6/2013