A Language for Adaptive Behaviors

Marco Tinacci

December 15, 2015

notations (distribution)

1 The proposed language

The structure of the proposed language naturally extends the AbC calculus suitably reduced in terms of operators and expressiveness. We consider AbC since it offers large expressiveness for defining communicating agents in an attribute-oriented fashion. We cut out communication of parameters that has been replaced by action labels to ease the understanding and to focus on the new features that the language offers.

citare

Precisare cosa si taglia e perche'

The language is intended for modeling adaptive agents that can interact with the surrounding environment through sensors and actuators. Main concepts of this language are *components*, that describe the agents that compose the whole system, and *processes*, that describe components' behaviors.

The original contribution of this language with respect to the state of the art, is the possibility to explicitly define an asymmetric relation "x observe y" using the *observe* operator $\triangleright^f$. In this way we say that component x is observing component y, in particular, using the filter function f, we can specify what can be perceived from y. Moreover the language gives the possibility to specify uncertainties on progression of processes and perception of observed signals from the considered environment.

We give a semantics in terms of partially observable models, in particular we consider nondeterminism, probabilistic behaviors and probabilistic perceptions. Since one of our target is computing schedulers for agents in order to reach a given target, the *hierarchical structure* imposed by operator $\triangleright^f$ allows a gradual resolution of the problem. This kind of compositional approach helps to avoid the state space explosion, indeed once a scheduler is computed for a small subproblem it can be applied reducing the dimension of the model, that also represents the environment of an higher problem.

citare POMDP

Syntax

The syntax of the language is reported in Table 1, where syntactical categories of *components* C, *processes* P, *process actions* α, *predicates* Π and *expressions*

E are described.

$$
\begin{array}{lcl}
C & ::= & \Gamma : P \mid C_1 || C_2 \mid C_1 \triangleright^f C_2 \\
P & ::= & \mathbf{0} \mid \Sigma_{i \in I} PP_i \mid K \\
PP & ::= & \langle \Pi \rangle \, \alpha \bigoplus_{i \in I} p_i : P_i \\
\alpha & ::= & (a)[\sigma] \mid (a)@\Pi[\sigma] \\
\Pi & ::= & \mathtt{tt} \mid E_1 \bowtie E_2 \mid \Pi_1 \wedge \Pi_2 \mid \ldots \\
E & ::= & v \mid n \mid this.n \mid \ldots
\end{array}
$$

Table 1: The syntax of the language

We describe now the intuitive meaning of each syntactical element, we use $Proc$ and $Comp$ to denote respectively the set of every process and every component generated by the proper syntactic category. Components (C) is a process P paired with its *attribute environment* Γ when it is expressed as $\Gamma : P$. The attribute environment is a partial function $\Gamma : \mathcal{A} \hookrightarrow \mathcal{V}$ that maps attribute names $\mathcal{A}$ into values $\mathcal{V}$. A component can also be a *parallel composition* of other components $C_1 || C_2$ or a component that observe another component $C_1 \triangleright^f C_2$. The *filter function* $f : Comp \to (\mathcal{A} \hookrightarrow \mathcal{V})$ is a function that return properly filtered information of the observed component C_2.

A process can assume one of the following structures: the *inactive process* $\mathbf{0}$, the *action-prefixed probabilistic process* $\langle \Pi \rangle \alpha \bigoplus_{i \in I} p_i : P_i$, the *nondeterministic choice* $\Sigma_{i \in I} P_i$ and the *recursive call* K. Action-prefixing is extended with a probabilistic evolution of the process, p_i is the probability that the process will evolve as P_i once action α is executed. Action-prefixing also integrates a predicate Π inherited from the original awareness process of AbC, it works as a guard that can disable the execution of the action α and the probabilistic evolution if it is not satisfied under the attribute environment Γ. When the guard is satisfied the execution of α and the following process are enabled and aware of Π. To ease the notation we can write $\langle \Pi \rangle \alpha (P_1 \oplus_{p_2} P_2)$ when $I = \{1, 2\}$ and $p_1 = 1 - p_2$. In the same way we can simplify the general notation of the non-deterministic choice $\Sigma_{i \in \{1,2\}} P_i$ with $P_1 + P_2$. Finally we use $K \triangleq P$ to define a process K that behaves like P and to enable recursive calls.

aggiornare a PP

actions, predicates, expressions, examples ...

Well-formedness To ease the understanding and the complexity of the language we give a "well-formedness" definition of terms generated by the syntax.

> **Definition 1** (Well-formed terms)**.** *A term generated by syntax in Table 1 is* well-formed *if all the following conditions hold:*

1. ***Probability distribution:***
   $P = \langle \Pi \rangle\ \alpha \bigoplus_{i \in I} p_i : P_i \Rightarrow \Sigma_{i \in I}\ p_i = 1 \wedge p_i \geq 0\ \forall\ i \in I$

2. ***Unique processes definitions:***
   $P = \langle \Pi \rangle\ \alpha \bigoplus_{i \in I} p_i : P_i \Rightarrow \exists!\ K \triangleq P_i,\ \forall i \in I$
   $P = \Sigma_{i \in I} P_i \Rightarrow \exists!\ K \triangleq P_i,\ \forall i \in I$

3. ***Absence of internal non-determinism:***
   $P = \Sigma_{i \in I} P_i \wedge P_l = \langle \Pi_l \rangle\ \alpha_l \bigoplus \cdots \wedge P_r = \langle \Pi_r \rangle\ \alpha_r \bigoplus \ldots$
   $\Rightarrow [\![\Pi_l]\!]_\Gamma = [\![\neg \Pi_r]\!]_\Gamma \vee \alpha_l \neq \alpha_r$

troppo restrittivo

spiegare punti delle condizioni

Operational semantics

Semantics of the language is given by means of transition relations $\dot{\mapsto}. \subseteq Proc \times PAct \times (\mathcal{A} \hookrightarrow \mathcal{V}) \times Dist(Proc)$ and $\dot{\rightarrow} \subseteq Comp \times CAct \times Dist(Comp)$ defined respectively in Table 2 and Table 3. $PAct$ and $CAct$ are respectively sets of actions at the process level and at the component level, the former is generated by the syntactic category α while the latter is the set Act extended to a actions to receive and $\overline{a}$ to send, formally $CAct = \{\overline{a} \mid a \in Act\} \cup Act$. Relation $\dot{\mapsto}.$ describes the probabilistic evolution of a process, while relation $\dot{\rightarrow}$ describes how this behavior is lifted to the component level.

definire insieme Act

$$(\textsc{psum})\ \frac{\Delta(P_k) = p_k \quad [\![\Pi]\!]_\Gamma \doteq \mathtt{tt}}{\langle \Pi \rangle\ \alpha \bigoplus_{i \in I} p_i : P_i \overset{\alpha}{\mapsto}_\Gamma \Delta}\ j, k \in I$$

$$(\textsc{sum})\ \frac{P_j \overset{\alpha}{\mapsto}_\Gamma \Delta}{\Sigma_{i \in I} P_i \overset{\alpha}{\mapsto}_\Gamma \Delta}\ j \in I \qquad (\textsc{rec})\ \frac{P \overset{\alpha}{\mapsto}_\Gamma \Delta \quad K \triangleq P}{K \overset{\alpha}{\mapsto}_\Gamma \Delta}$$

Table 2: Operational semantics for processes

Process semantics Rules in Table 2 describes the processes' behaviors, we use $\Delta \in Dist(Proc)$ to denote a distribution of processes.

Rule (PSUM) puts together guard evaluation, action prefixing and probabilistic choice. If the guard is satisfied then the process can evolve performing a process action α into a process P_i with the given probability p_i. Probabilities needs to respect Definition 1 in order to be well-formed and to guarantee a correct construction of a probability distribution.

Aggiornare semantica PSUM a PP

Rule (SUM) is the non-deterministic choice among a sumet of processes $\{P_i\}_{i \in I}$ that are enabled to perform some action. The process distribution

Δ that a process P would reach performing an action α is the same Δ that is reached when P is inside a sum and the same action α is executed. This is true in absence of internal non-determinism, that is one of the assumptions of well-formedness in Definition 1.

Rule (REC) is the standard recursive call. Since we assume that every process K has one and only one process definition $K \triangleq P$ we cannot have ambiguities in recursive calls.

Component semantics Rules in Table 3 describe the semantics of components and how to lift a process behavior to the component level, we use $\Theta \in Dist(Comp)$ to denote a distribution of components.

$$(\text{LIFT1})\ \frac{P \xmapsto{\Pi(a)[\sigma]}_\Gamma \Delta \quad \Gamma' = [[\sigma]]_\Gamma \quad \Theta(\Gamma' : P') = \Delta(P')}{\Gamma : P \xrightarrow{a} \Theta}$$

$$(\text{LIFT2})\ \frac{P \xmapsto{(a)@\Pi[\sigma]}_\Gamma \Delta \quad \Gamma' = [[\sigma]]_\Gamma \quad \Theta(\Gamma' : P') = \Delta(P')}{\Gamma : P \xrightarrow{\overline{a}} \Theta}$$

$$(\text{SYNC})\ \frac{C_1 \xrightarrow{a} \Theta_1 \quad C_2 \xrightarrow{a} \Theta_2}{C_1||C_2 \xrightarrow{a} \mathscr{P}(\Theta_1, \Theta_2)} \qquad (\text{RCV})\ \frac{C_1 \xrightarrow{\overline{a}} \Theta_1 \quad C_2 \xrightarrow{a} \Theta_2}{C_1||C_2 \xrightarrow{\overline{a}} \mathscr{P}(\Theta_1, \Theta_2)}$$

$$(\text{OBS1})\ \frac{C_1 \xrightarrow{a} \Theta_1 \quad C_2 \xrightarrow{a} \Theta_2}{C_1 \triangleright^f C_2 \xrightarrow{a} \mathscr{O}(\Theta_1, f, \Theta_2)} \qquad (\text{OBS2})\ \frac{C_1 \xrightarrow{\overline{a}} \Theta_1 \quad C_2 \xrightarrow{a} \Theta_2}{C_1 \triangleright^f C_2 \xrightarrow{\overline{a}} \mathscr{O}(\Theta_1, f, \Theta_2)}$$

Table 3: Operational semantics for components

SYNC e RCV possono essere modellate in termini di OBS1 e OBS2 usando una funzione filtro che nasconde tutto

Rules (LIFT1) and (LIFT2) are the basic rules that lift the process transition relation $\mapsto$ to the component transition relation $\rightarrow$. If a process P with attribute environment Γ executes an action and evolves into a process distribution Δ then the component $\Gamma : P$ can evolve into the component distribution Θ that is built by extending Δ with $\Theta(\Gamma' : P') = \Delta(P')$. When the action performed by the process P is of the form $\Pi(a)[\sigma]$ then the label of the component action is a, while when the action is of the form $(a)@\Pi[\sigma]$ the component action is the complementary, namely $\overline{a}$. Action a is intended as a *receive* and $\overline{a}$ as a *send*.

Rule (SYNC) is the synchronization on the same input action a of two processes that can receive the same action. Let $\Theta_1, \Theta_2 \in Dist(Comp)$ be two distributions of components, then $\mathscr{P}(\Theta_1, \Theta_2) \in Dist(Comp)$ is again a distribution of components built as follows

$$\mathscr{P}(\Theta_1, \Theta_2)(C) = \begin{cases} \Theta_1(C_1) \cdot \Theta_2(C_2) & \text{if } C = C_1||C_2 \\ 0 & \text{otherwise} \end{cases}$$

Rule (RCV) describe the communication between two components that perform complementary actions a and $\overline{a}$. The probabilistic evolution is represented by distribution $\mathscr{P}(\Theta_1, \Theta_2)$ as in rule (SYNC).

Rule (OBS) describe an observation made by the left-hand side component C_1 on the right-hand side component C_2. This happens when C_2, the observed environment, can perform an action a and evolve into a component distribution $\Theta \in Dist(Comp)$. The observation composition of C_1 and C_2 evolves then into the component distribution $\mathscr{O}(\Theta_1, f, \Theta_2) \in Dist(Comp)$ defined as follows (operator $\bullet : Comp \times (\mathcal{A} \hookrightarrow \mathcal{V})$ is defined in Table 4)

entrambe send e receive?

$$\mathscr{O}(\Theta_1, f, \Theta_2)(C) = \begin{cases} \Theta_1(C_1) \cdot \Theta_2(C_2) & \text{if } C = C_1 \bullet f(C_2) \rhd^f C_2 \\ 0 & \text{otherwise} \end{cases}$$

$$C \bullet \Gamma = \begin{cases} \Gamma' \circ \Gamma : P & \text{if } C = \Gamma' : P \\ C_1 \bullet \Gamma \mid\mid C_2 \bullet \Gamma & \text{if } C = C_1 \mid\mid C_2 \\ C_1 \bullet \Gamma \rhd^f C_2 \bullet \Gamma & \text{if } C = C_1 \rhd^f C_2 \end{cases}$$

Table 4: Inductive definition of an attribute environment applied to a component

2 Case studies

Swarm robotics A collection of robots have to detect the position of a victim of an accident. The idea to approach such a problem is to give the same behavior to every robot. The representative behavior is initialized consists in moving around the area at random until the victim is found, then the state switch to *rescuer*

citare paper yehia e paper margheri

Component $\Gamma_R : P_R$ represents a robot by means of its attribute environment Γ_R and its behavior defined by process P_R. The observed environment is represented by the single component $\Gamma_V : P_V$ that is the victim. The system can be defined as the parallel composition of many robots that observe the victim component, formally:

$$\Gamma_R : P_R || \ldots || \Gamma_R : P_R \rhd^f \Gamma_V : P_V$$

Every robot component knows its position that is stored in the local attribute *loc*. Local attribute *vPos* is used to store the victim's position and it is initially set to a *null* value to say that this information is not known yet.

$$
\begin{array}{rcl}
\Gamma_R & \triangleq & \{pos \leftarrow 2, victimPerceived \leftarrow \texttt{ff}\} \\
P_R & \triangleq & \overline{move}\ [this.pos \leftarrow next(this.pos)].P_R \\
& + & obs\ [this.victimPerceived \leftarrow \texttt{tt}, this.vPos \leftarrow fvPos].P_R \\
& + & \langle this.victimPerceived = \texttt{tt} \rangle P_{res} \\
P_{res} & \triangleq & \ldots \\
\Gamma_V & \triangleq & \{vPos \leftarrow 5\} \\
P_V & \triangleq & \overline{obs}.P_V
\end{array}
$$

$$
\begin{array}{rcl}
f(\{vPos, \ldots\} : P) & = & [1/3 : \{fvPos \leftarrow vPos\}, \\
& & 1/3 : \{fvPos \leftarrow vPos + 1\}, \\
& & 1/3 : \{fvPos \leftarrow vPos - 1\}]
\end{array}
$$

Cloud computing .

citare lavoro SCEL

$$\Gamma_g : S_g || \Gamma_s : S_s || \Gamma_b : S_b \rhd^f C_1 || \ldots || C_n$$

Components:

$$
\begin{array}{rcl}
S_g & \triangleq & \langle workload(g,s,b) < 40\% \rangle\ gReq[g \leftarrow g+1].S_g \\
& + & \langle workload(g,s,b) < 80\% \rangle\ sReq[s \leftarrow s+1].S_g \\
& + & \langle workload(g,s,b) < 90\% \rangle\ bReq[b \leftarrow b+1].S_g \\
& + & \overline{gServ}[g \leftarrow g-1].S_g \\
& + & \overline{sServ}[s \leftarrow s-1].S_g \\
& + & \overline{bServ}[b \leftarrow b-1].S_g \\
S_s & \triangleq & \ldots \\
S_b & \triangleq & \ldots \\
\Gamma_g & \triangleq & \{g \leftarrow 0, s \leftarrow 0, b \leftarrow 0\} \\
\Gamma_s & \triangleq & \{g \leftarrow 0, s \leftarrow 0\} \\
\Gamma_b & \triangleq & \{g \leftarrow 0\} \\
C_i & \triangleq & \emptyset : P_i \\
P_i & \triangleq & \tau.(p : \overline{gReq}.P_i, q : \overline{sReq}.P_i, 1-p-q : \overline{bReq}.P_i)
\end{array}
$$