Context-parameterized coordination model and semantics for behavioural programs

I. Nunes
Department of Informatics,
Faculty of Sciences, University of Lisbon
Lisbon, Portugal

Abstract. *In opposition to the traditional approaches to reactive systems in which the atomicity of actions is assumed, the behavioural paradigm deals with durative actions [9,10] in the sense that the results of a given action can interfere with other concurrently executing ones. This paradigm is, in its essence, a coordination model [6], which implies that the aspects of coordinationand computation are separated. This coordination is achieved through the use of guards, among other coordination primitives, that define not only the situations in which a given action is desired but also the ones in which its results can become obsolete. In this paper we present a coordination model for behavioural programs in which a given action can be considered obsolete not only at the time it finishes but also during its execution. This change in the semantics of the post-guard prevents the time-warp effect of obsolete actions, therefore allowing to simplify programs. We use a temporal logic where actions play a role to define an axiomatic semantics for behavioural programs.*

Keywords:*Durative actions, interference, coordination, contexts, temporal logic*.

1. Introduction

The *behavioural paradigm* developed in [9,10] for supporting multiprocessor computing was presented in [6] as a coordination model for controlling the execution of durative actions, i.e. actions which, although executed atomically on a private local state, have a duration in the sense that the system state in which they finish executing is not necessarily the same in which they started due to the interference of other concurrently executed actions.

In the proposed framework, each action of a *behavioural program* has two associated guards. On the one hand, a *pre-guard* that characterizes the states of the system in which the action is desirable. As long as there are free processors, all actions that should be launched are launched regardless of any possible future conflict. Conflicts are resolved at acceptance time. This is achieved through a *post-guard* that characterises the system states in which an already "locally-executed" action is acceptable (may update the system state).

The interface between the global space of the system and the local space in which each action executes is made through two atomic operations of *loading* – reading (part of) the global space – and *discharging* – updating (part of) the global space. The parts of the global space that are read and written, for each action, are defined by two lists of attributes: the loading and the discharging lists, respectively.

Like in (more) traditional coordination languages [1,2,3,4], the coordination model presented in [6] is responsible for controlling the interference between the actions and is independent from the computation model in which actions execute. That is to say, it does not really matter if the computations performed locally by the actions are programmed in a, say, imperative or functional style.

In the present paper we propose a coordination model for behavioural programs where the semantics of the post-guard is of a *continuation guard* instead of an acceptance one. Because actions are durative, the global state of the system may change during the execution of a given action due to the completion of other concurrently executing actions. The models we present here are based in computations or runs in which the post-guards of actions must hold during their whole execution, that is, the post-guard of a given action must hold in all system states that are established while that action is being executed.

This change brings a simplification for behavioural programs because the time warp caused by "lazy" executed actions is no longer a problem.

When the semantics of the post-guard is of an acceptance guard [6,7], it is evaluated (over the system state) only when the action finishes local execution. During action execution the need for that kind of action can cease and then begin once again (due to changes in the state of the system caused by the discharging of other concurrently executing actions). If the post-guard is true when the action finishes local execution then the action is accepted, no matter what went on in the system in the meanwhile. Programs have to be specified in order to prevent lazy actions from modifying the system state with obsolete values, therefore becoming more complicated.

The semantics we present here for the post-guard (of a continuation guard) prevents this, insofar as late actions would never try to discharge their results because they would abort execution as soon as the continuation guard that defines their usefulness becomes false.

The behavioural paradigm proposed in [9,10] was presented in [6] as a coordination model for reactive programs with durative actions and a suitable operational semantics was given.

We now present the models and an axiomatic semantics for *behavioural programs with continuation guards* (abbreviated to *C_behavioural programs*for simplicity). Models are built from computations of durative transition systems [6,7]; a temporal logic extended with action expressions is used to express the formulas for the axiomatic semantics.

2. Coordination in the behavioural paradigm

We formalize the notion of C_behavioural program for a given *signature* Θ=(At, Ac) where At defines the set of attribute symbols and Ac defines the set of action symbols (equipped with two mappings L,D: $Ac \rightarrow 2^{At}$ providing, for each action $g \in Ac$, its *loading* (L_g) and *discharging* (D_g) lists of attributes).

We denote by $\mathcal{L}(At)$ the propositional language of *state formulas* – $S ::= p \mid \neg S \mid S_1 \supset S_2$ ($p \in At$) – defined over At and use it to express guards.

The behavioural paradigm is not characterized by a specific syntax for its programs but, instead, by a set of primitives aiming at coordinating the interference between durative actions. The restrictions, or assumptions, on the nature of the languages that can be used for programming individual actions, constitute what we call a *programming context*. Contexts allow us to define the semantics in a parameterized way, obtaining the independence between the models of computation and coordination at the formal level.

Definition 2.1: An *operational context* for a signature Θ=(Ac,At) consists of a pair $\mathcal{T}_o=(\{PL_g|g\in Ac\}, \{(l_g,d_g,\xrightarrow{g})|g\in Ac\})$ where,

- PL_g is a programming language over a collection $At_g \subseteq At$ of (local) attributes;
- l_g and d_g give semantics to the *loading* and *discharging* operations; l_g picks part of the system state (the one defined by the loading list) and returns a local state; d_g picks a local state and returns part of the system state (the one defined by the discharging list);
- $\xrightarrow{g}$ is an operational semantics for commands in language PL_g; it picks a PL_g command and a local state and returns a local state ♦

A C_behavioural program is defined with respect to a given context:

Definition 2.2: A *C_behavioural program* for a signature Θ=(Ac,At) and a context $\mathcal{T}_o = (\{PL_g|g\in Ac\}, \{(l_g,d_g,\xrightarrow{g})|g\in Ac\})$, is a pair (I,BDY) where I is an initial condition and BDY assigns, to every action symbol $g\in Ac$, a triple (A_g,P_g,C_g) – a *double guarded action* – where:

- A_g and P_g are state formulas; the *pre-* and *continuation-guards* of action *g*, respectively;
- C_g is a program in the language PL_g. ♦

In order to illustrate the coordination model just described, consider the following C_behavioural program (the "dining philosophers"). For the sake of clarity each action is represented as

(pre-guard,continuation-guard) $\rightarrow$
[loading_list | program | discharging_list]

```
(LOT∧ROT∧¬H) // defines initial states
(¬H, ¬H)  → [  | think | H]
(H∧PR∧PL, H∧PR∧PL)  →
        [  | eat | H, PR, ROT, PL, LOT]
(H∧PL∧¬PR∧¬ROT, ¬ROT)  → [  | rel_L | PL, LOT]
(H∧PR∧¬PL∧¬LOT, ¬LOT)  → [  | rel_R | PR, ROT]
(H∧LOT, H∧LOT)  → [  | take_L | PL, LOT]
(H∧ROT, H∧ROT)  → [  | take_R | PR, ROT]
```

where the propositional symbols are used with the following meanings: H (the philosopher is hungry); LOT (the left fork is on the table); ROT (the right fork is on the table); PL (the philosopher possesses the left fork); PR (the philosopher possesses the right fork).

The intended semantics of the rel_L action specification, for example, is the following. When a philosopher is hungry (H), possesses the left fork (PL) and does not possess the right fork ($\neg$PR), and the right fork is not on the table ($\neg$ROT), he will release the left fork. If, while our philosopher is in the process of releasing his left fork, the philosopher at his right releases his left fork (our philosopher's right fork), therefore turning ROT to true, then our philosopher should abort the releasing action.

We have deliberately omitted the definition of the program executed locally by each action to remind that the behavioural paradigm is all about coordinating durative actions and not about programming their effects. This coordination is achieved through the pre and continuation-guards and by the loading and discharging lists. Naturally, these local programs have to be supplied for the C_behavioural program to be complete.

3. A Model of Coordination

The models we present for C_behavioural programs are sets of computations or runs that are obtained from *durative transition systems* [6,7]:

Definition 3.1: A *state* for a signature Θ=(At,Ac) – $\sigma:2^{At}$ – is a subset of the set At of attributes. An *initialized durative transition system* (*dts*) for a signature Θ=(Ac,At) is a triple $<\Sigma,\Sigma_0,\{\xrightarrow{g} | g\in Ac\}>$ where:

- Σ is a non-empty set (of states),
- Σ_0 is a non-empty set (of initial states),
- $\xrightarrow{g}:(\Sigma\times\Sigma)\rightarrow\Sigma$ for each action $g\in Ac$, is a partial function (on both arguments). ♦

The notation σ_Ω stands for the projection of state σ in set $\Omega\subseteq At$. The notation $\sigma'[\sigma_\Omega/\Omega]$ stands for the state that is equal to σ' except for the attributes in $\Omega\subseteq At$ whose values are given by σ_Ω.

State formulas as defined above are evaluated over states for a signature Θ, in the following way:

$\sigma \models_\Theta p$ iff $p\in\sigma$
$\sigma \models_\Theta \neg S$ iff $\sigma \not\models_\Theta S$
$\sigma \models_\Theta S_1\supset S_2$ iff $\sigma \models_\Theta S_1$ implies $\sigma \models_\Theta S_2$

A C_behavioural program defines a durative transition system where the set of transitions is composed, for each action in Ac, of the triples of states that characterize the double-guarded action. These transitions relate the three most important states of instances of action execution: the state in which it is launched (satisfying the pre-guard), the state in which, having finished execution, it is considered for discharging (satisfying the continuation-guard), and the state resulting from the discharge of its results.

Definition 3.2: The *dts defined* by program CBP = $(I,\{(A_g,P_g,C_g)|g\in Ac\})$ for context $\mathcal{T}_o$ = $(\{PL_g|g\in Ac\}, \{(l_g,d_g,\xrightarrow{g})|g\in Ac\})$ is such that

- for all $\sigma_0\in\Sigma_0$, $\sigma_0 \models_\Theta I$;
- $(\sigma,\sigma',\sigma'') \in \xrightarrow{g}$ iff $\sigma \models_\Theta A_g$, $\sigma' \models_\Theta P_g$, and $\sigma''=\sigma'[d_g(\xrightarrow{g}(C_g,l_g(\sigma_{Lg})))/D_g]$ ♦

Notice that the values of the attributes of the system – At – in the resulting state σ'' are completely determined. The values for attributes in the discharging list (D_g) are given through the discharging (d_g) of the results of the execution ($\xrightarrow{g}$) of the local program C_g over the values that were loaded at launching time ($l_g(\sigma_{Lg})$); the other attributes – in At\D_g – depend on "what went on" in the system while *g* was executing, that is, they keep the value they had in the accepting state (σ').

The triples (launching, accepting, resulting) of states define the effects of actions but do not account for the truth value of the continuation guard during action execution, that is, in states established between the launching and the accepting ones. The models for our C_behavioural programs have to account for this:

Definition 3.3: A *computation* of the *dts defined* by program $(I,\{(A_g,P_g,C_g)|g\in Ac\})$ is an infinite sequence of states $r=\sigma_0,\sigma_1,\sigma_2....$ that satisfies the following requirements:

Initiation: $\sigma_0\in\Sigma_0$;

Consecution: for all j>0, there are i<j and $g\in Ac$ s.t. $(\sigma_i,\sigma_{j-1},\sigma_j)\in\xrightarrow{g}$;

Continuation: for all i<j s.t.

r(i)= σ and

r(j-1)= σ' and

r(j)= σ'' for some $(\sigma,\sigma',\sigma'')\in\xrightarrow{g}$,

we have that for all $i\leq k<j$, $r(k)\models_\Theta P_g$.

where r(i) is the ith+1 state of a computation. ♦

The *continuation* requirement tells us that in every action execution – characterized by a triple of states (launching, accepting, resulting) — the continuation-guard must be verified in all states that are established between the launching and the resulting ones.

4. Axiomatic Semantics

The *axiomatic semantics* of a given C_behavioural program CBP is given by a set of formulas in a temporal logic; these formulas are sound with respect to the model composed by all computations of the *dts* defined by CBP for a given context.

The logic we use is a temporal logic as presented in [5] extended with two *action expressions*. A propositional version of this temporal logic defines *temporal formulas* for a signature Θ,

$$T ::= S \mid g \mid \overline{g} \mid \neg T \mid T_1\supset T_2 \mid \mathrm{O}T \mid T_1\,\mathcal{U}\,T_2 \mid T_1\,\mathcal{S}\,T_2$$

where S is a state formula and $g\in Ac$.

These formulas are evaluated over pairs *(computation,index)* for a *dts* $<\Sigma,\Sigma_0,\{\xrightarrow{g}|g\in Ac\}>$:

$(r,i)\models_\Theta S$ iff $r(i)\models_\Theta S$

$(r,i)\models_\Theta g$ iff

there are $(\sigma,\sigma',\sigma'')\in\xrightarrow{g}$ and j>i s.t.

$r(i)=\sigma$, $r(j-1)=\sigma'$ and $r(j)=\sigma''$

$(r,i)\models_\Theta \overline{g}$ iff

there are $(\sigma,\sigma',\sigma'')\in\xrightarrow{g}$ and j<i s.t.

$r(j)=\sigma$, $r(i-1)=\sigma'$ and $r(i)=\sigma''$

The temporal operators *next* (O), *until* ($\mathcal{U}$) and *since* ($\mathcal{S}$) are as defined in [5]. The usual abbreviations for tt, ff, $\wedge$, $\vee$, and $\equiv$ are used.

The semantics of the action expressions g and $\overline{g}$ reflect the non-atomicity of action execution because the launching state need not be contiguous to the accepting state in a computation or run; an undefined number of actions can discharge their results while the action is executing. On the other hand, the fact that the accepting and resulting states are always contiguous in a computation reflects the atomicity of the test-and-set operation, that is, the evaluation of the continuation guard in the accepting state and the discharging of the local results into the system global state are done in one single atomic step (nothing happens in between).

Just like the definition of C_behavioural programs requires a context that accounts for the specificities of the programming languages that are used for defining the programs of actions, the axiomatic semantics requires a context that allows us to relate properties of the local executions with properties of the global system. Such an axiomatic context needs to serve two purposes.

On the one hand, it has to provide the ability to infer properties of local executions. For that purpose, we consider that each language PL_g (restricted to attributes in $At_g\subseteq At$) has an associated modal logic LC_g. We will need the following auxiliary definition:

Definition 4.1: Let PL_g be a programming language for a set At_g of attributes. We define the PL_g *associated context logic* as the triple $(LC_g, \models_{\mathcal{M}g}, \vdash_{LCg})$ where:

- LC_g is a modal logic for $\mathcal{L}(At_g)$ that includes the modal formulas ([c]A) (the syntax of commands *c* in PL_g is irrelevant here);
- $\models_{\mathcal{M}g}$ is a semantic consequence relation where:

– $\mathcal{M}_g=(\Sigma_g,[\![\ ]\!]:(PL_g\times\Sigma_g\rightarrow\Sigma_g))$ where Σ_g is the set of states for At_g and $[\![\ ]\!]$ gives semantics to commands in PL_g;

– for all $\sigma_g\in\Sigma_g$,

$$\sigma_g\vDash_{\mathcal{M}g}[c]A \text{ iff } [\![c]\!]\,\sigma_g\vDash_{\mathcal{M}g}A;$$

- $\vdash_{LCg}$ is a syntatic consequence relation which is sound wrt the semantics. ♦

On the other hand, an axiomatic context has to provide us with means for relating local and global properties similar to the way the loading and discharging functions operate at the operational level. For this purpose, we rely on *mixed inference rules*:

Definition 4.2: Let At_1 and At_2 be sets (of attributes) and $f:2^{At_1}\rightarrow 2^{At_2}$. An *f-mixed inference* rule is a pair $<\Gamma,A'>$ where $\Gamma\subseteq\mathcal{L}(At_1)$ and $A'\in\mathcal{L}(At_2)$, which we denote by $\Gamma\vdash_f A'$. Such a mixed inference rule is said to be *sound* iff for all $\sigma_1\in 2^{At_1}$, $\sigma_1\vDash\Gamma$ implies $f(\sigma_1)\vDash A'$. ♦

When we want to relate properties of local executions with properties of the global system we have to relate, for a given action *g*, the local and global attributes. We have to relate global attributes of the loading list of *g* (L_g) with local attributes of *g* (At_g) in "pre-execution" situations and also to relate local attributes of *g* with the global attributes of the discharging list of *g* (D_g) in "post-execution" situations. The mixed inference rules that are of interest in order to define the axiomatic semantics of a program CBP are:

– the ones composed of a set of state formulas Γ_{Ig} in the language of the loading attributes (L_g), paired with a state formula A_g in the language of the local attributes (At_g);

– the ones composed of a set of state formulas Γ_g in the language of the local attributes (At_g), paired with a state formula A_{Dg} in the language of the discharging attributes (D_g).

Definition 4.3: An *axiomatic context* $\mathcal{T}_a$ for a signature $\Theta=(At,Ac)$ is a family of tuples $\{((LC_g,\vDash_{\mathcal{M}g},\vdash_{LCg}),lr_g,dr_g)|g\in Ac\}$ where

- $(LC_g,\vDash_{\mathcal{M}g},\vdash_{LCg})$ is a PL_g associated context logic;
- lr_g is a set of $(2^{L_g}\rightarrow 2^{At_g})$-mixed inference rules (*loading* rules);
- dr_g is a set of $(2^{At_g}\rightarrow 2^{D_g})$-mixed inference rules (*discharging* rules). ♦

We now give the formulas that constitute the axiomatic semantics of the program *dga*s. These are formulas expressing pre- and continuation-guards, frame conditions (stating that all attributes not in the discharging list of an action are not affected by that action) and, finally, local effects of actions.

Definition 4.4: Given an axiomatic context $\mathcal{T}_a$ = $\{((LC_g,\vDash_{\mathcal{M}g},\vdash_{LCg}),lr_g,dr_g)|g\in Ac\}$ for a signature $\Theta=(At,Ac)$, the axiomatic semantics of a *double guarded action* (A_g,P_g,C_g) consists of the following set Γ_g of formulas:

(1)$(g\supset(A_g\wedge P_g\,\mathcal{U}\,\overline{g}))$

(2) $(O\overline{g}\supset P_g\mathcal{S}g)$

(3)for all $p\notin D_g$, the formulas

$((O\overline{g}\wedge p)\supset Op)$ and

$((O\overline{g}\wedge\neg p)\supset O\neg p)$

(4)for all

$<\{B_1\},B_1'>\in lr_g$ and

$<\{B_2'\},B_2>\in dr_g$ s.t. $\vdash_{LCg}B_1'\supset[C_g]B_2'$,

the formula $(g\wedge B_1)\supset tt\,\mathcal{U}(\overline{g}\wedge B_2)$

The axiomatic semantics of a C_behavioural program CBP is the set $\cup_{g\in Ac}\Gamma_g$. ♦

The formulas in (1) and (2) say that if there is a durative transition through *g* then the pre-guard of *g* is true in the launching state and the continuation-guard is true during its execution. The formulas in (3) express frame conditions, that is, that in all states that result from the execution of *g*, all attributes not belonging to the discharging list of *g* keep the values they had in the acceptance state. The formulas in (4) express the locality conditions. More precisely, they express that, if i) B_1 holds when *g* is launched, ii) B_1 translates to the local property B_1' through the loading rules, iii) C_g establishes B_2' when executed in (local) states that satisfy B_1', and iv) B_2' translates to B_2 through the discharging rules, then B_2 holds in the state that results from the execution of *g*. That is, the effects of the execution of *g* over the attributes in the discharging list of *g* are determined by the execution of C_g.

Notice that when L_g and D_g are "mirrored" directly in At_g in the sense that the functions $(2^{L_g}\rightarrow 2^{At_g})$ and $(2^{At_g}\rightarrow 2^{D_g})$ just copy them to and from the local state of *g*, we can assume that L_g and D_g are subsets of At_g so that the formulas for the locality conditions simplify to $\{(g\wedge B)\supset tt\,\mathcal{U}(\overline{g}\wedge A)\ |$

$\vdash_{LCg}$ $B \supset [C_g]A\}$ where B and A are state formulas in $\mathcal{L}(L_g)$ and $\mathcal{L}(D_g)$ respectively.

In order to prove that the formulas that constitute the axiomatic semantics of a C_behavioural program CBP are sound with respect to the model composed of the computations of CBP as described above, we have to be sure that the axiomatic and operational contexts used are somehow related. Only in this way can we ensure that we are dealing with two different "points of view" of the same program.

Definition 4.5: An axiomatic context $\mathcal{T}_a = \{((LC_g, \vDash_{\mathcal{M}g}, \vdash_{LCg}), lr_g, dr_g) | g \in Ac\}$ and an operational context $\mathcal{T}_o = (\{PL_g | g \in Ac\}, \{(l_g, d_g, \xrightarrow{g}) | g \in Ac\})$ for a signature Θ are said to *agree* iff, for all $g \in Ac$:

- $\mathcal{M}_g = (\Sigma_g, \xrightarrow{g})$;
- lr_g is a set of *sound* l_g-mixed inference rules;
- dr_g is a set of *sound* d_g-mixed inference rules. ♦

For an axiomatic and an operational contexts to agree we have that the semantics for program C_g, in logic LC_g, is given by its operational semantics – $\xrightarrow{g}$ – in $\mathcal{T}_o$, and the rules lr_g and dr_g that allow us to account for the local/global relations must "agree" with the semantics given in $\mathcal{T}_o$ – l_g and d_g – for the loading and discharging operations.

Therefore, the rules in lr_g, $<\Gamma, A>$, must be such that, whenever the formulas $B_i \in \Gamma$ ($B_i \in \mathcal{L}(L_g)$) hold in σ_{Lg}, formula A holds in $l_g(\sigma_{Lg})$, that is, A holds in the local state obtained from applying the loading operation l_g to σ_{Lg}. The rules in dr_g, $<\Gamma, A>$, must be such that, whenever the formulas $B_i \in \Gamma$ ($B_i \in \mathcal{L}(At_g)$) hold in σ_g, formula A holds in $d_g(\sigma_g)$, that is, A holds in σ_{Dg} obtained from applying the discharging operation d_g to σ_g.

Proposition 4.6: Let $\mathcal{T}_a$ and $\mathcal{T}_o$ be an axiomatic and an operational contexts for a signature $\Theta = (At, Ac)$ which agree with each other. Given $CBP = (I, \{(A_g, P_g, C_g) | g \in Ac\})$, a program for Θ, the axiomatic semantics for CBP – $\cup_{g \in Ac} \Gamma_g$ – as defined in 4.4, is *sound* with respect to the model composed of the computations of CBP. ♦

Proof.

We have to prove that the following formulas are true in the set of all computations for the *dts* $<\Sigma, \Sigma_0, \{\xrightarrow{g} | g \in Ac\}>$ defined by CBP:

- $(g \supset (A_g \wedge P_g \mathcal{U} \overline{g}))$
- $(O\overline{g} \supset P_g \mathcal{S} g)$
- for all $p \notin D_g$, the formulas

  $((O\overline{g} \wedge p) \supset Op)$ and

  $((O\overline{g} \wedge \neg p) \supset O\neg p)$
- for all

  $<\{B_1\}, B_1'> \in lr_g$ and

  $<\{B_2'\}, B_2> \in dr_g$ s.t. $\vdash_{LCg} B_1' \supset [C_g] B_2'$,

  the formula $(g \wedge B_1) \supset tt\mathcal{U}(\overline{g} \wedge B_2)$

We only present the proof for the last one here.

Suppose (1) $<\{B_1\}, B_1'> \in lr_g$ and

(2) $<\{B_2'\}, B_2> \in dr_g$ and

(3) $\vdash_{LCg} B_1' \supset [C_g] B_2'$.

Because $\mathcal{T}_a$ and $\mathcal{T}_o$ agree and because $\vdash_{LCg_g}$ is sound wrt $\vDash_{\mathcal{M}g}$, we have that,

for all $\sigma_g \in \Sigma_g$,

$\sigma_g \vDash_{\mathcal{M}g} B_1'$ implies $\xrightarrow{g}(C_g, \sigma_g) \vDash_{\mathcal{M}g} B_2'$ (4)

We know that $<\{B_1\}, B_1'>$ is a sound l_g-mixed inference rule, that is,

for all σ_{Lg},

$\sigma_{Lg} \vDash_\Theta B_1$ implies $l_g(\sigma_{Lg}) \vDash_{\mathcal{M}g} B_1'$ (5)

and $<\{B_2'\}, B_2>$ is a sound d_g-mixed inference rule, that is,

for all σ_g,

$\sigma_g \vDash_{\mathcal{M}g} B_2'$ implies $d_g(\sigma_g) \vDash_\Theta B_2$ (6)

from (5) and (4) it comes that,

for all σ_{Lg},

$\sigma_{Lg} \vDash_\Theta B_1$ implies $\xrightarrow{g}(C_g, l_g(\sigma_{Lg})) \vDash_{\mathcal{M}g} B_2'$ (7)

from (7) and (6) it comes that,

for all σ_{Lg},

$\sigma_{Lg} \vDash_\Theta B_1$ implies $d_g(\xrightarrow{g}(C_g, l_g(\sigma_{Lg}))) \vDash_\Theta B_2$ (8)

Because $B_1 \in \mathcal{L}(L_g)$, we have

for all $\sigma \in \Sigma$,

$\sigma \vDash_\Theta B_1$ implies $d_g(\xrightarrow{g}(C_g, l_g(\sigma_{Lg}))) \vDash_\Theta B_2$ (9)

Suppose $(r,a) \vDash_\Theta (g \wedge B_1)$ (10)

By definition, from (10) we have

there are $(\sigma, \sigma', \sigma'') \in \xrightarrow{g}$ and $b > a$ s.t.

$r(a) = \sigma$, $r(b-1) = \sigma'$, $r(b) = \sigma''$ and $r(a) \vDash_\Theta B_1$ (11)

We have to prove that $(r,a) \vDash_\Theta tt\mathcal{U}(\overline{g} \wedge B_2)$, that is, that

there is $b \geq a$ s.t. $(r,b) \vDash_\Theta (\overline{g} \wedge B_2)$,

that is, that there is $b \geq a$ s.t.

there are $(\sigma, \sigma', \sigma'') \in \xrightarrow{g}$ and $c < b$ s.t.

$r(c) = \sigma$, $r(b-1) = \sigma'$, $r(b) = \sigma''$ and $(r,b) \vDash_\Theta B_2$

Because we have (11), it is true that

there is $b \geq a$ s.t. $r(b) = \sigma''$ (12)

and the *c* value can be *a*. We have still to prove $(r,b) \vDash_\Theta B_2$. Because of (9) and (10) we have:

$$d_g(\xrightarrow{g}(C_g,l_g(r(a)_{Lg})))\models_{\Theta}B_2 \qquad (13)$$

From (12) we have $r(b)=\sigma''$. By definition 3.2, $\sigma''=\sigma'[d_g(\xrightarrow{g}(C_g,l_g(\sigma_{Lg})))/D_g]$. Then, from (13), $r(b)\models_{\Theta}B_2$. Because $B_2\in\mathcal{L}(D_g)$, we have $(r,b)\models_{\Theta}B_2$.

End of proof.

Properties of C_behavioural programs can be proven departing from these formulas and from all the fully established knowledge that the use of a temporal logic brings us [5,8].

5. Further work

The composition of programs under the behavioural paradigm is being studied together with the issue of preservation of properties. We are also adressing the application of the behavioural formalisms to the specification of real-time systems as a means of fully covering the notion of duration in critical systems.

References

1. J.P.Banâtre and D.Le Métayer, "Programming by Multiset Transformation", *Communications* ACM16, 1 pp. 55-77, 1993.
2. P.Ciancarini, C.Hankin, "Coordination Languages and Models", LNCS 1061, Springer-Verlag, 1996.
3. D.Gelernter, "Generative Communication in Linda", ACM *Trans. Prog. Lang. Syst.* 7, 1, pp. 80-112, 1985.
4. D.Gelernter, N.Carriero, "Coordination Languages and their Significance", *Communications* ACM 35, 2, pp. 97-107, 1992.
5. Z.Manna, A.Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag 1991.
6. I.Nunes, J.L.Fiadeiro and W.M.Turski, "Coordinating Durative Actions", in Proc. COORDINATION'97, D.Garlan and D.Le Métayer (eds), Lecture Notes in Computer Science 1282, pp. 115-130, Springer-Verlag 1997.
7. I.Nunes, J.L.Fiadeiro and W.M.Turski, "A Modal Logic of Durative Actions", in H.Barringer et al (eds), Advances in Temporal Logic, pp. 299-317, Kluwer Academic Publishers, 2000.
8. C.Stirling, "Modal and Temporal Logics", in S.Abramsky, D.Gabbay and T.Maibaum (eds), *Handbook of Logic in Computer Science* 2, pp. 477-563, 1992.
9. W.M.Turski, "On Specification of Multiprocessor Computing", *Acta Informatica* 27, pp. 685-696, 1990.
10. W.M.Turski, "Extending the Computing Paradigm", *Structured Programming* 13, pp. 1-9, 1992.