Interaction of agents and environments

Alexander Letichevsky[1], David Gilbert[2]

[1] Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine
let@d105.icyb.kiev.ua
[2] Department of Computer Science, City University, London EC1V 0HB, UK
drg@cs.city.ac.uk

Abstract. A new abstract model of interaction between agents and environments considered as objects of different types is introduced. Agents are represented by means of labelled transition systems considered up to bisimilarity. The equivalence of agents is characterised in terms of an algebra of behaviours which is a continuous algebra with approximation and two operations: nondeterministic choice and prefixing. Environments are introduced as agents supplied with an insertion function which takes the behaviour of an agent and the behaviour of an environment as arguments and returns the new behaviour of an environment. Arbitrary continuous functions can be used as insertion functions, and we use functions defined by means of rewriting logic as computable ones. The transformation of environment behaviours defined by the insertion function also defines a new type of agent equivalence — insertion equivalence. Two behaviours are insertion equivalent if they define the same transformation of an environment. The properties of this equivalence are studied. Three main types of insertion functions are used to develop interesting applications: one-step insertion, head insertion, and look-ahead insertion functions.

Keywords: agents, behaviour, distribution, environments, interaction, semantics

1 Introduction

The majority of traditional theories of interaction including CCS [19], CSP [9], ACP [3], TLA [17], and more recent theories such as game semantics [2], and tile model [6], consider interaction between agents in the environment. However the notion of an environment is used implicitly or its elements are introduced as elements of process algebra expressions undistinguished from agent expressions. In those models where the environment is considered explicitly such as programs over shared memory or Linda based models, the notion of an environment is very special. In this paper we consider agents and environments as objects of different types. Agents are represented by means of labelled transition systems with divergence and termination, considered up to bisimilarity. The equivalence of agents is characterised in terms of an algebra of behaviours which is a two sorted (actions and behaviours) continuous algebra with approximation and two operations: nondeterministic choice and prefixing (like basic ACP). The notion

of an abstract agent can be introduced as a transition closed set of behaviours. All known compositions in various kinds of process algebras can be then defined by means of continuous functions over agents.

Environments are introduced as agents supplied with functions used for the insertion of other agents into these environments. An insertion function has two arguments: the behaviour of an agent and the behaviour of an environment. The value of an insertion function is a new behaviour of an environment. The notion of an environment gives the possibility of defining a new type of agent equivalence — insertion equivalence. Two behaviours are insertion equivalent if they define the same transformation of an environment. Most of the known equivalences for processes can be characterised as insertion equivalence.

In earlier publications [13, 14, 16] the model has been considered in the context of language representation. The generic (Parameterised) Action Language (AL), introduced there was considered as a general model of computation and interaction covering a wide class of nondeterministic concurrent programming languages. The interaction semantics of AL has been defined in terms of transformations of environment behaviours and has been used for the definition of a computational semantics as well. In [15] a new, more abstract model of interaction between agents and environments has been introduced. This paper generalises the approach of previous ones, allowing the use of arbitrary continuous functions for the definition of insertion of an agent into an environment.

Three main types of insertion functions are used to develop interesting applications: one-step insertion, head insertion, and look-ahead insertion functions. They are introduced by means of rewriting logic [10]. We study insertion equivalence for one-step insertion using algebraic representation of agents and proving congruence property for the main operations of behaviour algebra. The implementation of the model on a base of algebraic programming system APS is considered.

2 Preliminaries

2.1 Transition systems

Definition 1. (Park [5]) A transition system over a set of actions A is a set S of states with a transition relation $s \stackrel{a}{\to} s', \quad s, s' \in S, \quad a \in A$, and two subsets S_Δ and $S_\perp$ called correspondingly sets of terminal and divergent states.

The original definition of D.Park does not contains terminal and divergent states. The former is used for the definition of computational semantics of agents, and the later for introducing the approximation relation and the technically important construction of infinite objects from finite ones by passing to limits.

Definition 2. A binary relation $R \subseteq S \times S$ is called a partial bisimulation if for all s and t such that sRt and for all $a \in A$

- $s \in S_\Delta \Rightarrow t \in S_\Delta$

– $s \xrightarrow{a} s' \Rightarrow \exists t'.t \xrightarrow{a} t' \wedge s'Rt'$
– $s \notin S_\perp \Rightarrow (t \notin S_\perp \wedge (t \xrightarrow{a} t' \Rightarrow \exists s'.s \xrightarrow{a} s' \wedge s'Rs))$

This definition is a slight modification of the definition in [1]. A state s of a transition system S is called a *bisimilar approximation* of t denoted as $s \sqsubseteq_B t$ if there exists a partial bisimulation R such that sRt. Symmetric closure of partial bisimulation is a *bisimulation equivalence* denoted $s \sim_B t$. The definition of partial bisimulation can be easily extended to the case when R is defined as a relation between the states of two different systems, considering the disjoint union of their sets of states. Two transition systems are bisimilarily equivalent if each state of one of them is bisimilarily equivalent to some state of another.

We give some consequences from this definition in order to help the reader to understand it better. The divergent state without transitions approximates arbitrary other state. If s approximates t and t is convergent (not divergent) then s is also convergent, s and t have transitions for the same sets of actions and satisfy the same conditions as for usual bisimulation without divergence. Otherwise if s is divergent (and therefore so is t) the set of actions for which s has transitions is only included in the set of actions for which t has transitions, i.e. s is less defined than t.

2.2 Behaviour algebra

A behaviour algebra (or an algebra of behaviours) over an action set A is a continuous algebra [8] or an algebra with approximation (a poset with a minimal element and continuous operations [12]). It has two operations, the first being denoted by $+$ is an internal binary aci-operation (idempotent associative and commutative operation). This operation corresponds to nondeterministic choice. The second operation is prefixing $a.u$, a being an action, u being a behaviour. The minimal element of a behaviour algebra is denoted by $\perp$. The empty behaviour Δ performs no actions and usually denotes the successful termination of a (computational) process. The impossible behaviour 0 is the neutral element for nondeterministic choice. There is also the impossible (empty) action $\emptyset$ in A. The identities of a behaviour algebra are shown in Figure 1.

$$u + v = v + u$$
$$(u + v) + w = u + (v + w)$$
$$u + u = u$$
$$u + 0 = 0 + u = u$$

$$\emptyset.u = 0$$

Fig. 1. Relations of an algebra of behaviours

The approximation relation of the algebra of behaviours over A is a partial order which satisfies the relations presented in Figure 2.

$$\bot \sqsubseteq u$$

$$u \sqsubseteq v \Rightarrow u + w \sqsubseteq v + w$$

$$u \sqsubseteq v \Rightarrow a.u \sqsubseteq a.v$$

Fig. 2. Approximation for behaviours

If all relations of a behaviour algebra are consequences of those presented in Figure 1 and the approximation relation is a minimal partial order satisfying the relations in Figure 2 then this algebra is called a free algebra. The elements of the minimal (initial) sub-algebra $F_{\mathtt{fin}}(A)$ of a free behaviour algebra over A (i.e. a sub-algebra generated by the empty behaviour, the impossible behaviour and the bottom element) are called *finite behaviours*. All other behaviours (of a free behaviour algebra) are assumed to be the limits (least upper bounds) of the directed sets of finite elements. The free behaviour algebra which includes all such limits is denoted $F(A)$. It is defined uniquely up to a continuous isomorphism.

Note that in $F(A)$ the fixed point theorem is true, so we can use it for constructing new behaviours from already built ones by means of equations of the form $X = F(X)$, where X is a vector of variables and $F(X)$ is an algebraic functional, that is a functional constructed from variables and constants Δ and $\bot$ by means of nondeterministic choice and prefixing. An alternative approach is to consider $F(A)$ as a final coalgebra and use coinduction for reasoning and constructing behaviours [4].

Each behaviour $u \in F(A)$ can be represented in the form

$$u = \sum_{i \in I} a_i.u_i + \varepsilon \tag{1}$$

where a_i are different from impossible action, u_i are behaviours, I is a finite (for finite elements) or infinite set of indices, $\varepsilon = \Delta, \bot, \Delta + \bot, 0$ (*termination constants*). If all summands in the representation (1) are different then this representation is unique up to the associativity and commutativity of nondeterministic choice. A behaviour u is called *divergent* if $\varepsilon = \bot, \Delta + \bot$ and *convergent* otherwise. Note that u is always divergent for infinite I as a limit of finite divergent sums. Convergent infinite sums can be introduced by extending the notion of a finite element. Namely, termination constants, prefixed finite elements and arbitrary (finite or infinite) sums of finite elements are also considered as finite elements.

2.3 Behaviours and transition systems

For each state $s \in S$ of a transition system let us consider a behaviour $\mathtt{beh}(s) = u_s$ (of a system in a given state s) defined as a component of a minimal solution

of a system

$$u_s = \sum_{s \xrightarrow{a} s'} a_i.u_{s'} + \varepsilon_s \tag{2}$$

where termination constants ε_s are defined in Figure 3.

$$s \notin S_\Delta \cup S_\perp \Rightarrow \varepsilon_s = 0$$
$$s \in S_\Delta \setminus S_\perp \Rightarrow \varepsilon_s = \Delta$$
$$s \in S_\perp \setminus S_\Delta \Rightarrow \varepsilon_s = \perp$$
$$s \in S_\Delta \cap S_\perp \Rightarrow \varepsilon_s = \Delta + \perp$$

Fig. 3. Termination constants for the behaviour of a system in a given state

A set U of behaviours is called *transition closed* if from $a.u+v \in U$ and $a \neq \emptyset$ it follows that also $u \in U$. Each transition closed set U can be considered as a set of states of a transition system with transitions $a.u + v \xrightarrow{a} u, a \neq \emptyset$, the set of terminal states $U_\Delta = \{u = v + \Delta\}$ and divergent states $U_\perp = \{u = v + \perp\}$. Therefore the relations $\sqsubseteq_B$ and $\sim_B$ can be considered for behaviours as well as for the states of a transition system.

Theorem 3. *Let s and s' are states of a transition system, u and v are behaviours. Then:*

1. $s \sqsubseteq_B s' \Leftrightarrow u_s \sqsubseteq u_{s'}$;
2. $s \sim_B s' \Leftrightarrow u_s = u_{s'}$;
3. $u = v \Leftrightarrow u \sim_B v$.

In the following we shall use $\sim$ instead of $\sim_B$.

2.4 Compositions of behaviours

There are many useful compositions defined in concurrency theory as operations on processes or agents represented as transition systems. The majority of them preserve bisimilarity and can therefore be defined as operations on behaviours. Another useful property of these compositions is continuity. To define a continuous function over behaviours it is sufficient to define it on finite behaviours and extend to all others by passing to limits. Definitions in the style of SOS semantics [20] or employing conditional rewriting systems always produce continuous functions. In this section two main compositions – sequential and parallel – will be defined.

Sequential composition of behaviours. u and v is a new behaviour denoted as $(u; v)$ and defined by means of the inference rules and equations presented in Figure 4.

$$u \xrightarrow{a} u' \vdash (u;v) \xrightarrow{a} (u';v)$$

$$(\Delta;u) = (u;\Delta) = u,\ (0;u) = 0,\ (\bot;u) = \bot$$

Fig. 4. Sequential composition of behaviours

In the following we shall also use the notation uv instead of $(u;v)$ and (au) instead of $(a.u)$. This notation is not ambiguous if we identify an action a with the behaviour $a.\Delta$.

Parallel composition of behaviours. Up to now the set of actions A was considered as a flat set without any structure. Now we define an algebraic structure on this set introducing the combination $a \times b$ of actions a and b. This operation is commutative and associative with the empty action as annulator ($a \times \emptyset = \emptyset$). Thus the set A becomes an algebra of actions.

The inference rules and equations for the definition of the parallel composition $u\|v$ of behaviours u and v are presented in Figure 5. Commutativity and associativity of parallel composition are consequences of this definition.

$$\frac{u \xrightarrow{a} u',\ \ v \xrightarrow{b} v',\ \ a \times b \neq \emptyset}{u\|v \xrightarrow{a \times b} u'\|v'}$$

$$u \xrightarrow{a} u' \vdash u\|v \xrightarrow{a} u'\|v,\ \ u\|(v+\Delta) \xrightarrow{a} u'$$

$$v \xrightarrow{a} v' \vdash u\|v \xrightarrow{a} u\|v',\ \ (u+\Delta)\|v \xrightarrow{a} v'$$

$$(u+\Delta)\|(v+\Delta) = (u+\Delta)\|(v+\Delta) + \Delta$$

$$(u+\bot)\|v = (u+\bot)\|v + \bot$$

$$u\|(v+\bot) = u\|(v+\bot) + \bot$$

Fig. 5. Parallel composition of behaviours

3 Agents and environments

The previous section contains fairly standard definitions and constructions which are used as the mathematical foundation of concurrency theory. Our approach is close to that of ACP [3], and we use the continuous algebra of behaviours as a domain for the characterisation of transition systems up to bisimilarity instead of power-domains as in [1] or [18]. In this section we introduce the main construction of our theory, namely the insertion of an agent into an environment.

An *abstract agent* U over an action algebra A is a transition closed set of behaviours over A. An agent can be initialized by distinguishing the set $U_0 \subseteq U$ of possible initial states so that each other state of an agent is reachable from some of the initial states.

Usually agents are represented by a transition systems and are identified with these systems. In this case the corresponding abstract agent is the set of all behaviours of the states of its representation. Two representations of the same agent are therefore bisimilarily equivalent.

The set of behaviours of an agent can be considered as a transition system as well (the standard representation of an agent) and we can speak about the set of states when considering the behaviours of an agent. We should distinguish between an agent as a set of states or behaviours and an agent in a given state. In the latter case we consider each individual state or behaviour of an agent as the same agent in a given state.

An *Environment* E is an agent over an *environment algebra of actions* C with an *insertion function*. The insertion function $\mathtt{Ins}$ of an environment is a function of two arguments: $\mathtt{Ins}(e, u) = e[u]$. The first argument e is a behaviour of an environment, the second is a behaviour of an agent over an action algebra A in a given state u (the action algebra of agents can be a parameter of an environment). An insertion function is an arbitrary function continuous in both of its arguments. The result is a new behaviour of the same environment.

For the definition of insertion functions we can use the same methods as for the definition of operations over behaviours, but the semantics of agents is different. They are considered up to an equivalence which is in general weaker than bisimilarity. This is *insertion equivalence* which depends on an environment and its insertion function. Two agents (in given states) or behaviours u and v are *insertion equivalent* with respect to an environment E, written $u \sim_E v$ if for all $e \in E$ $e[u] = e[v]$. Each agent u defines the transformation $\mathtt{Tr}_u^E : E \to E$ of its environment: $\mathtt{Tr}_u^E(e) = e[u]$ and $u \sim_E v$ iff $\mathtt{Tr}_u^E = \mathtt{Tr}_v^E$. We shall also use the notation $[u]$ for $\mathtt{Tr}_u^E$.

After inserting an agent into an environment, the new environment can accept new agents to be inserted, and the insertion of several agents is something that we will often wish to describe. We shall use the notation

$$e[u_1, \ldots, u_n] = e[u_1] \ldots [u_n]$$

for the insertion of several agents.

Note that in this expression $u_1, \ldots, u_n$ are agents inserted into the environment simultaneously, but the order can be essential for some environments. If you want agent u to be inserted after agent v, you must compute some transition $e[u] \xrightarrow{a} s$ and consider expression $s[v]$. Some environments can move independently, suspending the movement of an agent inserted into them. In this case if $e[u] \xrightarrow{a} e'[u]$ then $e'[u, v]$ describes the simultaneous insertion of v and u into the environment in a state e' as well as the insertion of u at the moment when an environment is in state e and after this the insertion of v.

An environment $e[u]$ with containing an inserted agent u can be used for the insertion of another agent using the insertion function $\mathtt{Ins}$, or can be considered

as a new agent which can be inserted into a new environment e' with another insertion function $\mathtt{Ins}'$. In this case $e'[e[u]] = \mathtt{Ins}'(e', \mathtt{Ins}(e, u))$, and we can associate with the behaviour u not only transformation $\mathtt{Tr}_u^E$ but also a function $F = \mathtt{Tr}_u^{E \times E' \to E'} : E \times E' \to E'$ defined by equation $F(e, e') = e'[e[u]]$.

In the sequel the notation $e[u]$ will be used not only for the case when u and e are behaviours (or expressions which take values in the behaviour algebra) but also states of transition systems used to represent corresponding behaviours. In this case we must prove the correctness of an expression, or its independence from the representation of a state, that is $e \sim e' \Rightarrow e[u] \sim e'[u]$.

Let us now consider some important cases of environments and insertion functions.

3.1 Parallel and sequential environments

The insertion function for a parallel environment is

$$e[u] = e \| u$$

In this case all agents inserted into an environment interact in parallel and $e[u_1, \ldots, u_n]$ does not depend on the order of insertion.

Another important case is a sequential environment:

$$e[u] = eu$$

In this case the performance of agents is sequential.

If $\Delta \in E$ then the insertion equivalence of agents is a bisimulation. A weaker equivalence can be obtained if the definition of the insertion function is modified in the following way:

$$e[u] = \varphi(e \| u)$$

for a parallel environment or

$$e[u] = \varphi(eu)$$

for a sequential one. In this modification φ is an arbitrary continuous transformation of E. The restriction function of CCS or the hiding function of CSP or their combinations are useful special cases of φ.

3.2 One-step insertion

The class of one-step insertion functions consists of insertion functions that define the interaction between environment and inserted agents in such a way that the current observable action of a resulting environment depends on the behaviour of an environment and agents in the current moment of time only (one-step behaviour). This dependency is defined by means of a *hiding function* $h : A \times C \to 2^C$ (in [16] the similar function was called a residual function). The formal definition is presented in Figure 6. In this figure ε_u is a termination constant in the canonical representation of $u = \sum a_i.u_i + \varepsilon_u$, ε is an arbitrary termination constant.

$$\frac{u \xrightarrow{a} u',\ e \xrightarrow{c} e',\ d \in h(a,e)}{e[u] \xrightarrow{d} e'[u']}$$

$$e \xrightarrow{c} e' \vdash e[u] \xrightarrow{c} e'[u]$$

$$e[u+\Delta] = e[u+\Delta] + e,\ e[u+\perp] = e[u+\perp] + e \| \perp,\ (e+\perp)[u] = e[u] + \perp$$

$$\varepsilon[u] = \varepsilon \| \varepsilon_u$$

Fig. 6. One-step insertion function

In order to prove the properties of one-step insertion it is useful to introduce its algebraic representation. Let us consider the canonical forms of the state (behaviour) $e = \sum_{i \in I} c_i.e_i + \varepsilon_e$ of an environment and the state $u = \sum_{j \in J} a_j.u_j + \varepsilon_u$ of an agent. The following representation of $e[u]$ is a consequence of its definition in Figure 6:

$$e[u] = \sum_{d \in h(a_j, c_i)} d.e_i[u_j] + \sum_{i \in I} c_i.e_i[u] + \beta(\varepsilon_u, e) \tag{3}$$

where $\beta(\varepsilon + \varepsilon', e) = \beta(\varepsilon, e) + \beta(\varepsilon', e),\ \beta(0, e) = 0 \| \varepsilon_e,\ \beta(\Delta, e) = e,\ \beta(\perp, e) = e \| \perp$. This representation provides the computation of prefixing and nondeterministic choice:

$$e[a.u] = \sum_{d \in h(a, c_i)} d.e_i[u] + \sum_{i \in I} c_i.e_i[a.u] + \beta(0, e) \tag{4}$$

$$e[u+v] = e \times [u] + e \times [v] + \sum_{i \in I} c_i.e_i[u+v] + \beta(\varepsilon_u, e) + \beta(\varepsilon_v, e) \tag{5}$$

where

$$e \times [u] = \sum_{d \in h(a_j, c_i)} d.e_i[u_j]$$

The equations (4) and (5) show that transformations $[a.u]$ and $[u+v]$ can be expressed in terms of $[u]$ and $[v]$ (as a minimal fixed point). Thus one-step insertion equivalence is a congruence (with respect to prefixing and nondeterministic choice) and these equations can be used for the definition of prefixing $a.[u] = [a.u]$ and nondeterministic choice $[u] + [v] = [u+v]$ on the set of continuous transformations of E. As a result the mapping $u \to [u]$ is a homomorphism.

A natural special case of a one-step insertion environment is a *memory* over some set R of names or variables. A state of this environment is a mapping $e : D^R \to D^R$. Actions $c \in C$ correspond to statements over R such as (parallel) assignments and conditions. If c is a statement then $e \xrightarrow{c} e'$ is a functional relation on E, and if c is a condition then $e \xrightarrow{c} e$ iff c is true on e. A combination over the set of actions $c \times c'$ can be defined as an action equivalent to the simultaneous performance of c and c'. In this case $c \times c' \neq \emptyset$ iff c and c' are consistent. Consistency can be defined for the synchronous or asynchronous combination of

actions, and for synchronous combination consistency means that each of two statements c and c' change the same variables. For asynchronous combination a stronger condition is used: neither of two statements can use the variables changed by the other one.

A hiding function h for a memory environment can be defined in the following way: $h(a,c) = \{d | c = a \times d\}$, if $a \neq c$ and $h(a,a) = \{\delta\}$, where δ is a special atomic action (empty statement) such that $\delta \times a = a \times \delta = a$ for an arbitrary action a and $e \stackrel{\delta}{\to} e$. A memory environment extended by input/output and interface statements can be used for modeling (deterministic or nondeterministic) sequential imperative programs over shared memory.

A useful extension of one-step insertion can be obtained by introducing tools for making some of the interactions of agents and environments unobservable. For this purpose let us introduce a special symbol o to denote the unobservable action and let $h : A \times C \to C \cup \{o\}$. Define the unlabeled transitions on the set of states $e[u]$:

$$\frac{u \stackrel{a}{\to} u',\ e \stackrel{c}{\to} e',\ o \in h(a,c)}{e[u] \to e'[u']}$$

and the rule:

$$\frac{e[u] \stackrel{*}{\to} e'[u'],\ e'[u'] \stackrel{d}{\to} e''[u'']}{e[u] \stackrel{d}{\to} e''[u'']}$$

A one-step environment with these two extra rules is called an *extended* one-step environment. For this environment a summand $\sum_{o \in h(a_j, c_i)} e_i[u_j]$ must be added to representation (3) and the congruence properties for the operations of behaviour algebra are still valid.

3.3 Head insertion

When we study the interaction of a client and a server the latter can be considered as the main part of an environment into which several clients can be inserted. An environment in this case can observe only the current action of a client (query, message, pushing buttons and so on). At the same time the server knows its internal state and can make a decision by analysing its future behaviour. This situation can be captured by head insertion.

A head insertion function is defined by means of three systems of rewriting rules. The rules of the first system have the form

$$(a, G(x_1, x_2, \ldots)) \to (d, G'(x_1, x_2, \ldots))$$

where $a \in A,\ d \in C$, $G(x_1, x_2, \ldots)$ and $G'(x_1, x_2, \ldots)$ are terms of a behaviour algebra over C with variables $x_1, x_2, \ldots$ considered up to the identities of this algebra. The relation defined by this system is called the *interaction move* and is denoted by $(a, e) \stackrel{\texttt{interact}}{\to} (d, e')$ The rule for this relation is:

$$\frac{(a, G(x_1, x_2, \ldots)) \to (d, G'(x_1, x_2, \ldots))}{(a, G(e_1, e_2, \ldots)) \to (d, G'(e_1, e_2, \ldots))}$$

The rules of the second system have the form

$$(a, G(x_1, x_2, \ldots)) \to G'(x_1, x_2, \ldots)$$

They define the *hidden move* relation which is denoted as $(a, e) \stackrel{\texttt{hidden}}{\to} e'$ The rules of the third system have the form

$$G(x_1, x_2, \ldots) \to (d, G'(x_1, x_2, \ldots))$$

They define the *environment move* which is denoted as $(a, e) \stackrel{\texttt{env}\bot\texttt{move}}{\to} (d, e')$.

$$\frac{u \stackrel{a}{\to} u',\ (a, e) \stackrel{\texttt{interact}}{\to} (d, e')}{e[u] \stackrel{d}{\to} e'[u']}$$

$$\frac{u \stackrel{a}{\to} u',\ (a, e) \stackrel{\texttt{hidden}}{\to} e'}{e[u] \to e'[u]}$$

$$\frac{e \stackrel{\texttt{env}\bot\texttt{move}}{\to} (d, e')}{e[u] \stackrel{d}{\to} e'[u],\ e[u + \Delta] \stackrel{d}{\to} e',\ e[u + \bot] \stackrel{d}{\to} e' \| \bot}$$

$$\frac{e[u] \stackrel{*}{\to} e'[u'],\ e'[u'] \stackrel{d}{\to} e''[u'']}{e[u] \stackrel{d}{\to} e''[u'']}$$

$$\frac{e[u] \stackrel{d}{\to} e'[u'] \vee e[u] \stackrel{*}{\to} e'[u'],\ f \sqsubseteq e,\ f[u] \not\to}{f[u] = \bot}$$

$$\varepsilon[u] = \varepsilon \| \varepsilon_u$$

Fig. 7. Head insertion function

The rules for transitions of $e[u]$ are presented in Figure 7. They include the unlabeled transitions defined by the hidden moves. An expression of the type $s \not\to$ means that there is no transitions $s \stackrel{d}{\to} s'$ or $s \to s'$.

The insertion function defined by the rules of Figure 7 is continuous. In order to prove this statement note that the knowledge of all finite approximations of e and u is sufficient for computing the transition $e[u] \stackrel{d}{\to} e'[u']$.

3.4 Look-ahead insertion

A more general situation in comparison with head insertion occurs if an environment contains the interpreter for some programming language and an agent is a software agent written in this language. In this case an environment can analyse not only its own future behaviour but the behaviour of an interpreted

program as well. This situation can be described by means of look-ahead insertion. This function is also defined by means of rewriting rules of only one type — interaction rules

$$(F(x_1, x_2, \ldots), G(y_1, y_2, \ldots)) \to (d, F'(x_1, x_2, \ldots), G'(x_1, x_2, \ldots))$$

These rules define an interaction relation denoted as

$$(u, e) \stackrel{\texttt{interact}}{\to} (d, u', e')$$

It can be proved that this general type of rewriting rules also covers hidden and environment moves (if we admit the possibility of an infinite number of rules which may be required to implement the transitive closure of unlabeled transitions).

$$\frac{(F(x_1, x_2, \ldots), G(y_1, y_2, \ldots)) \stackrel{\texttt{interact}}{\to} (d, F'(x_1, x_2, \ldots), G'(y_1, y_2, \ldots))}{G(e_1, e_2, \ldots)[F(u_1, u_2, \ldots)] \stackrel{d}{\to} G'(e_1, e_2, \ldots)[F'(u_1, u_2, \ldots)]}$$

$$\frac{e[u] = G(e_1, e_2, \ldots)[F(u_1, u_2, \ldots)] \stackrel{d}{\to} G'(e_1, e_2, \ldots)[F'(u_1, u_2, \ldots)),\ v \sqsubseteq u,\ f \sqsubseteq e,\ f[v] \not\to}{f[v] = \bot}$$

$$\varepsilon[u] = \varepsilon \| \varepsilon_u$$

Fig. 8. Look-ahead insertion function

The rules for look-ahead insertion function are presented in Figure 8. A look-ahead insertion function is also continuous; the proof is the same as that for a head insertion.

3.5 Distributed environments

We can obtain multilevel distributed structures using recursive insertion and different environments used on different levels. Let E_1 be some environment used as a local environment shared by several agents (shared memory or constraint store, for instance). An environment $e[u_1, \ldots, u_n]$ can be closed by applying to it some continuous function φ and changed to an agent which can be inserted to the environment E_2 of the next level. Several agents $v_1, \ldots, v_m$ constructed this way can be inserted to E_2 and a new environment $e[v_1, \ldots, v_m]$ can be considered as a distributed environment with local components (environments) $v_1, \ldots, v_m$. This construction can be repeated recursively. Look-ahead insertion can use the low level insertion function for computation of transitions of low level components.

4 Insertion equivalence

In this section we shall study a one-step equivalence. First the notion of normalised behaviour representation will be introduced and the criteria of one-step insertion equivalence of agents will be established. Then we shall study the congruence properties of sequential and parallel composition of agents.

Let E be a one-step environment with a hiding function h. First, let us note that if $u \sim_E v$ that is $[u] = [v]$ then $[au + bv] = [(a + b)u]$. This relation is also valid for an infinite number of summands:

$$[\sum_{i \in I} a_i.u_i] = [(\sum_{i \in I} a_i)u]$$

if all u_i are equivalent to u. A behaviour which is a sum of actions will be called a *one-step behaviour*. An arbitrary behaviour can be represented up to equivalence (wrt E) as a sum

$$\sum_{i \in I} p_i u_i + \varepsilon \tag{6}$$

where p_i are one-step behaviours and $[u_i] \neq [u_j]$ if $i \neq j$. To obtain this representation for the behaviour $\sum_{i \in I} a_i.u_i + \varepsilon$ it is sufficient to partition all summands $a_i.u_i$ collecting together those of them for which u_i are mutually equivalent and apply the equation above.

Let us extend the hiding function to one-step behaviours by defining for $p = \sum_{i \in I} a_i$, $h(p, c) = \bigcup_{i \in I} h(a_i, c)$ and $h(p) = \bigcup_{c \in C} h(p, c)$. For a one-step behaviour p if $h(p) = \emptyset$ then $[pu] = [0]$ and $[pu + v] = [v]$. Therefore, the representation (6) can be restricted so that for all $i \in I$ $h(p_i) \neq \emptyset$. Such a representation is called a *normal form* of an agent for the environment E.

Definition 4. A one-step environment is called *regular* if:

1. For all $a \in A$ and $c \in C$ $c \notin h(a, c)$;
2. E is a subalgebra of $F(C)$.

One-step behaviours p and q are called equivalent wrt a hiding function h ($p \sim_h q$) if for all $c \in C$ $h(p, c) = h(q, c)$. If p and q are equivalent then $[pu] = [qu]$.

Theorem 5. *For a regular one-step environment the normal form of a behaviour is unique up to the commutativity of nondeterministic choice and equivalence of the one-step behaviour coefficients.*

To prove the theorem let us first prove that if $h(p) \neq \emptyset$ and $[pu] = [qv]$ then $p \sim_h q$ and $[u] = [v]$ (the inverse is evident). Let $d \in h(p)$, then for some $a \in A$ and $c \in C$ $d \in h(a, c)$. Let us take an arbitrary state (behaviour) $e \in E$. Since E is an algebra, $ce \in E$. We have $c \neq d$ therefore $(ce)[pu] \stackrel{d}{\to} e[u]$. From the equivalence of pu and qv it follows that $(ce)[pu] \sim (ce)[qv] \Rightarrow (ce)[qv] \stackrel{d}{\to} e[v]$ and this is the only transition from $(ce)[qv]$ labelled by d. Therefore $d \in h(q)$ and $e[u] \sim e[v]$. From the arbitrariness of e we have $[u] = [v]$. Symmetric reasoning gives also $d \in h(q) \Rightarrow d \in h(p) \Rightarrow p \sim_h q$.

Next we show that if $u = \sum_{i \in I} p_i u_i + \varepsilon_u,\ v = \sum_{j \in J} q_j v_j + \varepsilon_v$ are two normal forms and $[u] = [v]$ then for each $i \in I$ there exists $j \in J$ such that $[p_i u_i] = [q_j v_j]$ and from symmetry these forms are the same up to the commutativity and equivalence of coefficients. Again, as above if $(ce)[p_i u_i] \stackrel{d}{\to} e[u_i], c \neq d$ there exists only one j such that $(ce)[q_j v_j] \stackrel{d}{\to} e[v_j]$ and vice versa. Therefore $p_i \sim_h q_j,\ [u_i] = [v_j]$ and $[p_i u_i] = [q_j u_j]$. The equality of ε_u and ε_v is obvious. □

Sequential composition has a congruence property for regular one-step environments, as shown by the following theorem.

Theorem 6. *Let E be a regular one-step environment. Then* $[u] = [u'] \wedge [v] = [v'] \Rightarrow [uv] = [u'v']$.

To prove this theorem we prove that the relation $e[uv] \sim_R e[u'v']$ defined for an arbitrary $e \in E,\ u, v, u', v' \in F(A)$ by the condition $[u] = [u'] \wedge [v] = [v']$ is a bisimilarity. In order to compute transitions, normal forms for the representation of agent behaviours must be used. We omit the details of this proof for reasons of space.

Parallel composition does not in general have a congruence property. To find the condition when it does, let us extend the combination of actions to one-step behaviours assuming that

$$p \times q = \sum_{p=a+p',\ q=b+q'} a \times b$$

The equivalence of one-step behaviours is a congruence if $h(p) = h(q) \Rightarrow h(p \times r) = h(q \times r)$.

Theorem 7. *Let E be a regular one-step environment and the equivalence of one-step behaviours is a congruence. Then* $[u] = [u'] \wedge [v] = [v'] \Rightarrow [u \| v] = [u' \| v']$.

As for the previous theorem we prove that the relation $e[u \| v] \sim_R e[u' \| v']$ is defined for an arbitrary $e \in E,\ u, v, u', v' \in F(A)$ by the condition $[u] = [u'] \wedge [v] = [v']$ is a bisimilarity. To compute transitions, normal forms for the representation of agent behaviours must be used as well as the algebraic representation of parallel composition:

$$u \| v = u \times v + u \between v + v \between u$$

The details are also omitted.

5 Implementation

The model described in the paper has been implemented in algebraic programming system APS [11] based on rewriting logic. The Action Language has been used as a language for the description of agents. The main compositions in the Action Language (AL) are nondeterministic choice, parallel and sequential compositions. Actions are considered as primitive statements. The syntax and semantics of combinations and other operations in the algebra of actions are

parameters of AL which is considered as a generic model for a class of nondeterministic concurrent programming languages. Procedure calls are another kind of primitive statement. The syntax of these kind of statements is also a parameter of AL as well as their intensional semantics which is defined by means of the *unfold operator* represented in the form of a rewriting system (recursion). The intensional semantics of a program is defined as the behaviour of an agent, and the interaction semantics is a parameter of a language and defined by means of rewriting rules for the insertion function for a given environment.

The language also has the possibility to describe variables and localising them within local program components which can be used for the description of distributed agents. Variables are considered as variables of a memory state or a constraint store considered as a local environment for agents, and the meaning of a local component is an agent inserted into its local environment. The parallel composition of local components is considered as a set of agents inserted into the higher level environment which is a shared memory or a shared constraint store.

The first implementation of AL by means of an interpreter written in APLAN (the source language of APS) has been described in [7]. The next step was the development of a simulator which has been used to study the semantics of concurrent constraint and probabilistic concurrent constraint languages [21]. These early implementations used one-step insertion only. The current implementation is based on head insertion and can be easily extended to look-ahead insertion.

The simulator is an interactive program which can explore the behaviour of an environment with agents inserted into it step-by-step, with branching at nondeterministic points and return back to previous states. In automatic mode it can search for states with specific properties, such as successful termination or dead-lock states.

6 Conclusions

A model of interaction between agents and environments based on insertion functions has been presented in this paper. The set of behaviour transformations has been introduced as a domain for the semantic description of agents inserted into a corresponding environment. This description reflects the interaction of agents and environments and mathematically is represented by a continuous mapping from behaviours to transformations. For a regular one-step insertion this mapping is a continuous homomorphism.

The model has been implemented in the algebraic programming system APS and this implementation is being used to study interaction and computation in declarative programming paradigms.

References

1. S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92(2):161–218, 1991.

2. Samson Abramsky. Semantics of interaction. In *Trees in Algebra and Programming – CAAP'96, Proc. 21st Int. Coll., Linköping*, volume 1059 of *Lecture Notes in Computer Science*, page 1. Springer-Verlag, 1996.
3. J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.
4. B.Jacobs and J.Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the EATCS*, 62:222–259, 1997.
5. D.M.R.Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conf*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.
6. U.Montanari F.Gadducci. The tile model. Technical Report TR-96-27, Department of Computer Science, University of Pisa, 1996.
7. D. R. Gilbert and A. A. Letichevsky. A universal interpreter for nondeterministic concurrent programming languages. In Maurizio Gabbrielli, editor, *Fifth Compulog network area meeting on language design and semantic analysis methods*, Sep 1996.
8. I. Guessarian. *Algebraic semantics*. Lecture Notes in Computer Science v.99. Springer-Verlag, 1981.
9. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, UK, 1985.
10. J.Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.
11. J. V. Kapitonova, A. A. Letichevsky, and S. V. Konozenko. Computations in aps. *Theoretical Computer Science*, 119:145–171, 1993.
12. A. A. Letichevsky. Algebras with approximation and recursive data structures. *Kibernetika*, (5):32–37, September-October 1987.
13. A. A. Letichevsky and D. R. Gilbert. Toward an implementation theory of nondeterministic concurrent languages. Technical Report 1996/09. ISSN 1364-4009, Department of Computer Science, City University, 1996. Also presented at the Second workshop of the INTAS-93-1702 project Efficient Symbolic Computing St Petersburg, Russia, October 1996.
14. A. A. Letichevsky and D. R. Gilbert. A general theory of action languages. Technical report, Department of Computer Science, City University, London, UK, Aug 1997.
15. A. A. Letichevsky and D. R. Gilbert. Agents and environments. In *1st International scientific and practical conference on programming, Proceedings 2-4 September, 1998*. Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine, 1998.
16. A. A. Letichevsky and D. R. Gilbert. A general theory of action languages. *Cybernetics and System Analysis*, (1):16–36, January-February 1998.
17. L.Lamport. The temporal logic of actions. *acm Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.
18. G. Milne and R. Milner. Concurrent processes and their syntax. *J. Assoc. Comput. Mach.*, 26(2):302, 1979.
19. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
20. G. Plotkin. A structured approach to operational semantics. Technical Report Tech.Rep. DAIMI FN-19, Computer Science Dept., Aarhus University, 1981.
21. T. Valkevych, D.R. Gilbert, and A.A. Letichevsky. A generic workbench for modelling the behaviour of concurrent and probabilistic systems. In *Workshop on Tool Support for System Specification, Development and Verification, at TOOLS98*, Malente, Germany, June 2–4, 1998.

This article was processed using the LaTeX macro package with LLNCS style