Agents and environments

Alexander Letichevsky
Glushkov Institute of Cybernetics
National Academy of Sciences of Ukraine
let@d105.icyb.kiev.ua

David Gilbert
Department of Computer Science
City University, UK
drg@cs.city.ac.uk

October 8, 1998

Abstract

A new abstract model of interaction between agents and environments is introduced. It can be used as a semantical framework for studying interaction as well as for the definition of inetraction semantics of Action Language studied in the previous paper of the authors. An interaction of agents and environments is defined in a more strong and more simmetric way in comparison with this publication. The basic notion used in the paper is the well known notion of a transition system with divergency and termination. For the definition of various transition systems constructed in the paper three main forms: recursive algebraic definitions, equational definitions, and structured operational semantics are used.

1 Introduction

In [6] a general model of computation and interaction has been introduced. This model was presented as a generic (Parametrised) Action Language (AL) which covers a wide class of models of interaction. 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. This paper introduces a new and more abstract model of interaction between agents and environments. It can be used as a semantical framework for studying interaction as well as for the definition of inetraction semantics of AL. An interaction of agents and environments is defined in a more strong and more simmetric way in comparison with [6]. The basic notion used in the paper is the well known notion of a transition system with divergency and termination. For the definition of various transition systems we use three main forms: recursive algebraic definitions, equational (rewriting logic [4], ACP [2]) definitions, and (logic) definitions in the form of inference rules of structured operational semantics (SOS [7]). The last form gives the explicite definition of a transition relation and is also called a transition definition.

2 Transition systems

Definition 1 ([3]) *A transition system over a set of actions A is a set S of states with a transition relation $s \xrightarrow{a} s'$, $s, s' \in S$, $a \in A$ and two subsets S_Δ and $S_\perp$ called correspondingly sets of terminal and divergent states.*

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 s' denoted as $s \sqsubseteq_B s'$ if there exists a partial bisimulation R such that sRs'. Symmetric closure of partial bisimulation is a *bisimulation* equivalence denoted $s \sim_B s'$.

3 Behaviour algebra

A behaviour algebra (or an algebra of behaviours) over an action set A is an algebra with approximation (poset with a minimal element and continuous operations [5]). It has two operations, the first being denoted by $+$ is an internal binary aci-operation (idempotent ac-operation). This operation corresponds to nondeterministic choice. The second operation is prefixing au, a being an action, u being a behaviour. The minimal element of a behaviour algebra is denoted $\perp$. The empty behaviour Δ performs no actions and usually denotes the termination of a process. The impossible behaviour 0 is the neutral element for nondeterministic choice. There is also the impossible action 0 in A denoted in the same way as impossible behaviour. The operations of a behaviour algebra must satisfy the identities shown in Figure 1. The symbol 0 in the left hand side of the last relation is an impossible action (not impossible behaviour).

$$\begin{gathered}u + v = v + u\\(u + v) + w = u + (v + w)\\u + u = u\\u + 0 = 0 + u = u\\[1em]0u = 0\end{gathered}$$

Figure 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.

$$\begin{gathered}\perp \sqsubseteq u\\[0.5em]u \sqsubseteq v \Rightarrow u + w \sqsubseteq v + w\\[0.5em]u \sqsubseteq v \Rightarrow au \sqsubseteq av\end{gathered}$$

Figure 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 this algebra is called a free one. The elements of the minimal (initial) sub-algebra $F_{\mathtt{fin}}(A)$ of a free behaviour algebra over A that is 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 countable 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.

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 nonzero actions, u_i are behaviours, I is a finite (for finite elements) or infinite (but countable) set of indices, $\varepsilon = \Delta, \perp, \Delta + \perp, 0$ (*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.

4 Behaviours and transition systems

For each state $s \in S$ of a transition system let us consider a behaviour u_s (of a system in a given state) 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 a Figure 3.

$$s \notin S_\Delta \cup S_\perp \Rightarrow \varepsilon_s = 0$$

$$s \in S_\Delta \setminus S_\perp \Rightarrow \varepsilon_s = \Delta$$

$$s \in S_\perp \setminus S_\Delta \Rightarrow \varepsilon_s = \perp$$

$$s \in S_\Delta \cap S_\perp \Rightarrow \varepsilon_s = \Delta + \perp$$

Figure 3: Termination constants for the behaviour of a system in a given state

A set U of behaviours is called *transition closed* if from $au+v \in U$ and $a \neq 0$ 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 $au + v \xrightarrow{a} u, a \neq 0$, the set of terminal states $U_\Delta = \{u = v + \Delta\}$ and divergent states $U_\perp = \{u = v + \perp\}$.

It can be proved that $s \sqsubseteq_B s' \Leftrightarrow u_s \sqsubseteq u_{s'}$, $s \sim_B s' \Leftrightarrow u_s = u_{s'}$, and $u = v \Leftrightarrow u \sim_B v$.

5 Sequential composition of behaviours

Three representations of definition will be considered.

1. Transition representation.

$$u \xrightarrow{a} u' \Rightarrow uv \xrightarrow{a} u'v$$

$$u \xrightarrow{a} u' \Rightarrow \Delta u \xrightarrow{a} \Delta u'$$

Terminal states $\Delta(v + \Delta)$, divergent states $(u+ \perp)v$, $\Delta(v+ \perp)$.

2. Algebraic definition (for $F(A)$ only).

$$uv = (\sum a_i u_i + \varepsilon_u)v = \sum a_i(u_i v) + \varepsilon_u v$$

$$\Delta u = u\Delta = u, 0u = 0, \perp u = \perp$$

3. Equational definition.

$$(u + v)w = uw + vw$$

$$(au)v = a(uv)$$

$$\Delta u = u\Delta = u$$

$$0u = 0$$

$$\perp u = \perp$$

All three definitions are equivalent for $F(A)$.

6 Parallel composition of behaviours

Now the combination of actions $a \times b$ is defined on a set of actions, so this set is an algebra (of actions). The combination of actions is supposed to be a commutative and associative with impossible action as annulator ($a \times 0 = 0$).

1. Transition definition (Figure 4).

$$\frac{u \xrightarrow{a} u', \;\; v \xrightarrow{b} v', \;\; a \times b \neq 0}{u\|v \xrightarrow{a\times b} u'\|v'}$$

$$u \xrightarrow{a} u' \Rightarrow u\|v \xrightarrow{a} u'\|v$$

$$v \xrightarrow{a} v' \Rightarrow u\|v \xrightarrow{a} u\|v'$$

Figure 4: Transition system representing parallel composition

Terminal states: $u\|v$ is terminal if $u = u' + \Delta$ and $v = v' + \Delta$. Divergent states: $u\|v$ is divergent if $u = u'+ \perp$ or $v = v'+ \perp$.

2. Algebraic definition (Figure 5).

3. Equational definition (Figure 6).

All three definitions are equivalent for $F(A)$, parallel composition is associative and the extension of behaviour algebra by equations of third definition is conservative.

$$u = (\sum a_i u_i + \varepsilon_u)$$

$$v = (\sum b_j v_j + \varepsilon_v)$$

$$u\|v = \sum (a_i \times b_j)(u_i\|v_j) + \sum a_i(u_i\|v) + \sum b_j(u\|v_j) + \alpha(\varepsilon_u, v) + \alpha(\varepsilon_v, u) = \texttt{expand}(u\|v)$$

$$\alpha(0, z) = 0$$

$$\alpha(\Delta, z) = z$$

$$\alpha(\perp, z) = \perp$$

$$\alpha(\Delta + \perp, z) = z + \perp$$

Figure 5: Algebraic definition of parallel composition

7 Interaction of agents with an environment

An abstract agent U over an action algebra A is a transition closed set of behaviours (states) over A. An agent can be initialized by picking out 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 ones).

An environment is a tuple $\langle E, A, C, \texttt{res}\rangle$ where E is a transition closed subset of behaviour algebra $F(C)$ over an algebra of action C called the behaviour algebra of an environment, and

$$\texttt{res} : C \times A \to 2^{C\setminus\{0\}}$$

or equivalently

$$\texttt{res} \subseteq C \times A \times C \setminus \{0\}$$

is called a *residual function (or relation).* The set E is also called a set of behaviour states of an environment, and its symbol sometimes is used as a symbol of an environment instead of a four-tuple.

Each state $u \in U_0$ of an agent U defines a continuous function $u[.] : E \to F(C)$ on the set of states of an environment E which describes the interaction of an agent with an environment. Three definitions of this function are the following.

1. Transition definition. State of a transition system is an expression $u[v]$. Presented in the Figure 7. A state $u[v]$ is terminal if $u = u' + \Delta$ and $v = v' + \Delta$ and divergent if $u = u' + \perp$ or $v = v' + \perp$.

2. Algebraic definition. Figure 8.

3. Equational definition (Figure 9). The function `interact` is defined in the following way.

$$\texttt{interact}(a, v) = \{(d, v') | \exists (c, v''). \ v = cv' + v'' \wedge c \xrightarrow{a} d\}$$

A function $(u, v) \to u[v]$ describes the insertion of an agent U in a state u to the environment E in a state v. This operation can be used in two ways:

1. To build a new agent W with the set of states $\{u[v]\}$ over C. This new agent can be later inserted to another environment.

$$u \| v = u \times v + u \lfloor\!\lfloor v + v \lfloor\!\lfloor u$$

$$x \times y = y \times x$$

$$(x + y) \times z = x \times z + y \times z$$
$$(ax) \times (by) = (a \times b)(x \| y)$$
$$\varepsilon \times x = 0$$

$$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$$
$$ax \lfloor\!\lfloor y = a(x \| y)$$
$$\varepsilon \lfloor\!\lfloor x = \alpha(\varepsilon, x)$$

Figure 6: Equational definition of parallel composition

$$\frac{u \xrightarrow{a} u', \quad v \xrightarrow{c} v', \quad c \xrightarrow{a} d}{u[v] \xrightarrow{d} u'[v']}$$

$$\frac{v \xrightarrow{c} v', \quad c \xrightarrow{0} d}{u[v] \xrightarrow{d} u[v']}$$

$$\frac{u \xrightarrow{a} u', \quad 0 \xrightarrow{a} d}{u[v] \xrightarrow{d} u'[v]}$$

Figure 7: Interaction of an agent with an environment, transition representation

2. To build a new environment with the same set of states and use this environment for the insertion of another agents.

8 Action Language

Syntactical presentation of agents.

$\texttt{Prog} ::= \ \texttt{Act} \mid \texttt{TermConst} \mid \texttt{ProcCall} \mid (\texttt{Prog} + \texttt{Prog}) \mid (\texttt{Prog} \| \texttt{Prog}) \mid (\texttt{Prog}; \texttt{Prog}) \mid$
$\quad \texttt{Loc}(\texttt{SetVar}, \ \texttt{Prog})$

`TermConst` are some of termination constants (at least Stop for Δ should be used). We distinguish between simple programs which contains no `Loc` construction, and programs with local progrm components that is constructions of the type $\texttt{Loc}(X, P)$. Intensional semantics of a simple program is an agent which can be obtained by means of unfolding procedure calls and defining transitions on a set of program states similar to [6] (Figure 10 presents this agent as

$$u = (\sum a_i u_i + \varepsilon_u)$$

$$v = (\sum b_j v_j + \varepsilon_v)$$

$$u[v] = \sum_{d \in \mathtt{res}(b_j, a_i)} d(u_i[v_j]) + \sum_{d \in \mathtt{res}(0, a_i)} d(u_i[v]) + \sum_{d \in \mathtt{res}(b_j, 0)} d(u[v_j]) + \varepsilon_u[\varepsilon_v]$$

$$\Delta[\varepsilon] = \varepsilon, \quad \perp [\varepsilon] = 0[\perp] = \perp, \quad 0[\Delta] = 0[0] = 0$$

Figure 8: Interaction of an agent with an environment, algebraic representation

a transition system considering (unfolded) programs up to assciativity and commutativity of nondeterministic choice and parallel composition and assciativity of sequential composition). Interaction semantics of simple programs can be defined for program agents as above.

To define an intensional semantics of a program with local components one must extend the set of actions to show explicitely local variables actions depend on. Extended actions will be denoted as $\mathtt{act}\ X(a)$ where $X \subset \mathtt{Var}$, $a \in \mathtt{Act}$. Now to define intensional semantics the reductions and transitions of Figure 11 must be added. Here σ is a protected renaming, Y and Z are correspondingly changed sets of local variables.

8.1 A store

To define an interaction semantics for programs with local components we introduce the notion of a store over a set of variables V and actions A (V-A-store). It is a tuple

$$\langle S, V, A, \mathtt{Fvar}, \mathtt{Ren}_S, \mathtt{Ren}_A, T \rangle$$

where

- S is the set of states of a store;
- V is the set of variables;
- A is the set of actions (with zero action 0);
- $\mathtt{Fvar} : S \cup A \to V$, is a function which computes the set of free variables for states and actions;
- $\mathtt{Ren}_S : S \times \Sigma \to S$ and $\mathtt{Ren}_A : A \times \Sigma \to A$ are the renaming functions for S and A correspondingly, where Σ is the set of one-to-one transformations of V (renaming substitutions or renamings);
- T is the transition relation $T \subseteq S \times A \times S$.

The set S of states of a store is a transition system over A with transition relation T and we write $s \xrightarrow{a} s'$ for $(s, a, s') \in T$. We also write $s\sigma$ ($a\sigma$) for $\mathtt{Ren}_S(s, \sigma)$ ($\mathtt{Ren}_A(a, \sigma)$). All attributes of a store must satisfy the following axioms:

- $\sigma \in \Sigma,\ s \xrightarrow{a} s' \Rightarrow s\sigma \xrightarrow{a\sigma} s'\sigma$,

$$
\begin{aligned}
u[v] &= u \times [v] + u \lfloor\lfloor [v] + [v] \lfloor\lfloor u \\
(u+u') \times [v] &= u \times [v] + u' \times [v] \\
(ax) \times [v] &= \sum_{(d,v') \in \mathtt{interact}(a,v)} d(u[v']) \\
\varepsilon \times [v] &= u \times [\varepsilon] = 0 \\
(u+u') \lfloor\lfloor [v] &= u \lfloor\lfloor [v] + u' \lfloor\lfloor [v] \\
au \lfloor\lfloor [v] &= \sum_{d \in \mathtt{res}(0,a)} d(u[v]) \\
\Delta \lfloor\lfloor [\varepsilon] &= \varepsilon \\
\perp \lfloor\lfloor [\varepsilon] &= 0 \lfloor\lfloor [\perp] = \perp \\
0 \lfloor\lfloor [\Delta] &= 0 \lfloor\lfloor [0] = 0 \\
[v+v'] \lfloor\lfloor u &= [v] \lfloor\lfloor u + v' \lfloor\lfloor u \\
[bv] \lfloor\lfloor u &= \sum_{d \in \mathtt{res}(b,0)} d(u[v]) \\
[\varepsilon] \lfloor\lfloor u &= 0
\end{aligned}
$$

Figure 9: Interaction of an agent with an environment, equational definition

- $(\forall x \in \mathtt{Fvar}(y)\ \sigma(x) = \sigma'(x)) \Rightarrow y\sigma = y\sigma'$.

The definition of a store covers the notions of memory (state is a function from V to data domain), constraint store (states are sets of constraints of some constraint system considered up to the equivalence), data and knowledge bases.

8.2 Programs over store

A store is a special type of an (internal) environment of a program and interaction of a program with this environment defines a program agent over store. The states of an environment are the states of a store and the function `interact` is defined in the following way:

$$\mathtt{interact}(\mathtt{act}\ X(a), [s]) = \{h(X, s, a, s') | s \xrightarrow{a} s'\}$$

where $[s]$ is the behaviour of a store in a state s, h is a hiding function which means that $\mathtt{Fvar}(h(X, s, a, s')) \cap X = \emptyset$.

Transition definition of this agent is presented in the Figure 12 (we assume that $\mathtt{res}(0, a) = \mathtt{res}(a, 0) = \emptyset$). We write $\mathtt{Loc}(X, P, s)$ for $(\mathtt{Loc}(X, P))[s]$.

An action program inserted to a local store as an environment is considered as an agent which later can be inserted into an external environment to define a global interaction semantics of a program.

$$a \xrightarrow{a} \Delta$$

$$\frac{P \xrightarrow{a} Q}{P + R \xrightarrow{a} Q, (P;R) \xrightarrow{a} (Q;R), P\|S \xrightarrow{a} Q\|S}$$

$$\frac{P \xrightarrow{a} Q, P' \xrightarrow{a'} Q', a \times a' \neq 0}{P\|P' \xrightarrow{a \times a'} Q\|Q'}$$

Figure 10: Intensional semantics of simple programs

$$\mathtt{Loc}(X,P) + Q \rightarrow \mathtt{Loc}(Y, P\sigma + Q)$$

$$\mathtt{Loc}(X,P)\|Q \rightarrow \mathtt{Loc}(Y, P\sigma\|Q)$$

$$(\mathtt{Loc}(X,P);Q) \rightarrow \mathtt{Loc}(Y,(P\sigma;Q))$$

$$\mathtt{Loc}(X,\mathtt{Loc}(Y,P)) \rightarrow \mathtt{Loc}(X \cup Z, P\sigma)$$

$$P \rightarrow Q \Rightarrow \mathtt{Loc}(X,P) \rightarrow \mathtt{Loc}(X,Q)$$

$$P \xrightarrow{a} Q \Rightarrow \mathtt{Loc}(X,P) \overset{\mathtt{act}\ X(a)}{\rightarrow} \mathtt{Loc}(X,Q)$$

$$\frac{P \xrightarrow{*} Q, \quad Q \xrightarrow{a} R}{P \xrightarrow{a} R}$$

Figure 11: Reductions and transitions of local program components

9 Examples

1. Let $C = A$, $a \xrightarrow{b} a \times b$ if $a \times b \neq 0$, $0 \xrightarrow{a} a$, and $a \xrightarrow{0} a$. Then

$$P[\Delta] = P$$

$$P[Q[\Delta]] = P\|Q$$

2. Let $C = A \cup \{e\}$, $e \xrightarrow{a} a$, $\mathtt{res(a,b)} = \emptyset$ if $a \neq e$. Let $v_0 = ev_0$. Then

$$P[v_0] = Pv_0$$

$$Q[P[v_0]] = PQv_0$$

3. To get CCS one can define renaming and restriction as procedure calls, use Milner action algebra for combination ($a \times \overline{a} = \tau, a \times b = 0$ if $b \neq \overline{a}$), and apply to an environment of the first example. Unfolding relations for the restriction are:

$$(P + Q) \setminus L = P \setminus L + Q \setminus L$$

$$(P\|Q) \setminus L = \mathtt{expand}(P\|Q) \setminus L$$

$$\frac{\texttt{Loc}(X,P) \overset{\texttt{act}\ Y(a)}{\rightarrow} \texttt{Loc}(Y,Q), \quad s \xrightarrow{a} t}{\texttt{Loc}(X,P,s) \overset{h(Y,s,a,t)}{\rightarrow} \texttt{Loc}(Y,Q,t)}$$

Figure 12: Program agent over store

$$(aP) \setminus L = a(P \setminus L), \quad a, \overline{a} \notin L$$

$$(aP) \setminus L = 0, \quad a \in L \vee \overline{a} \in L$$

Unfolding relations for renaming:

$$(P+Q)[f] = P[f] + Q[f]$$

$$(P \| Q)[f] = P[f] \| Q[f]$$

$$(aP)[f] = f(a)(P[f])$$

Note that there is no sequential composition in CCS. If P is a procedure call then

$$P \setminus L = \texttt{unfold}(P) \setminus L$$

$$P[f] = \texttt{unfold}(P)[f]$$

4. To get π-calculus one should use a store with substitutions for variables as states.

References

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

[2] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.

[3] 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.

[4] J.Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[5] A. A. Letichevsky. Algebras with approximation and recursive data structures. *Kibernetika*, (5):32–37, September-October 1987.

[6] A. A. Letichevsky and D. R. Gilbert. A general theory of action languages. *Kibernetika*, (1):16–36, January-February 1998.

[7] G. Plotkin. A structured approach to operational semantics. Technical report, Tech.Rep. DAIMI FN-19, Computer Science Dept., Aarhus University, 1981.