PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 42, NO. 1, PP. 49-70 (1998)

BROADCAST AUTOMATA: A COMPUTATIONAL MODEL FOR MASSIVELY PARALLEL SYMBOLIC PROCESSING

Monica ALDERIGHI*, Fabio CASINI**, Riccardo Pietro Giovanbattista MAZZEI** and Giacomo Renato SECHI*

* Istituto di Fisica Cosmica e Tecnologie Relative
Consiglio Nazionale delle Ricerche
Via Bassini 15 I-20133 Milano, Italy
Tel: +39 2 2369-9332, Fax: +39 2 236-2946
Email: {monica,giacomo}@ifctr.mi.cnr.it
** Dipartimento di Fisica
Universita' degli Studi di Milano
Via Celoria 21
I-20133 Milano, Italy

Received: September 30, 1997

Abstract

This paper presents a suitable formalism for the Broadcast Automata System, a model of massively parallel computation, introduced by the authors for prototyping of scientific applications. The model consists of a collection of identical entities, modelled as finite state automata, a global synchroniser providing coordination between the automata and a broadcast communication system, to which each automaton is connected, granting information exchange among the automata. The formalism is based on an extension of the classical formalism for finite state automata. The application to a case study concerning the recognition of first order propositional formulae is illustrated and the correctness proof is sketched.

Keywords: system models, parallel systems, correctness, finite automata.

1. Introduction

The Broadcast Automata based System (BAS) was defined as a model of massively parallel computation based on broadcast agents (ALDERIGHI et al., 1997). According to a reference architecture, it consists of a collection of identical agents, modelled as finite state automata (SA), a global synchroniser providing coordination between the agents and a broadcast communication system, to which each agent is connected, granting information exchange among the agents.

The number of automata simultaneously operating is not *a priori* fixed but can be varied as the complexity of the application changes. The topology is regular, and it is a fully connected graph in a more specifical way.

Therefore each SA is a neighbour of every other SA and can observe its state, yet it is not aware of its identity.

Computation in the BAS occurs as global evolution of all automata. In analogy with dynamic systems, it can be viewed as passing from a highly unstable state to more stable ones, until highly stable states (final states) are reached. These may correspond to a legal termination of the computation (the highest stability) or to an anomalous functioning condition.

Different formalisms and methodologies for distribute systems have been proposed in last years, such as Petri nets (REISIG, 1985), CCS (MILNER, 1980), CSP (HOARE, 1985), I/O-automata (JONSSON, 1989), Statecharts (HAREL, 1987), UNITY (CHANDY et al, 1988), FOCUS (DEDERICHS et al, 1993) and many other variants and integrations. The BAS is distinguished from these approaches for its emphasis on the identical structure of its components, which makes the BAS closer to the models of massive parallelism than to models of parallelism in general. Within this context, broadcast automata may be regarded as cellular automata (GARZON, 1995), yet they are distinguished from cellular automata, and classical finite state automata, for two respects: a sequential aspect in the state transition diagram and the concept of *elective affinity* (ALDERIGHI et al, 1997). This last expresses a specialised dependence of a BA state transition function on the state of other BAs and defines a relationship of state sensitivity among BAs: a BA is sensitive to its affined only. Elective affinity allows to dynamically define logical connections among BAs that are not physically connected, thus providing a sort of privileged communication among BAs.

The possible combinatorial state explosion resulting from complex applications is a not minor drawback of the use of this model. It is possible to rely upon abstraction processes over state components, for instance introducing *typed* states. Moreover thinking in terms of feasible systems, it may be argued that although the concept of observation is theoretically expressive and powerful, it can hardly be implemented efficiently. This drawback could be overcome by introducing events, which asynchronously notify BAs about significant state changes only. The topology of the communication subsystem possibly reduces the observability of the BAs and also increases system's reconfigurability costs, in terms of extra communication loads needed to properly rearrange connections among BAs. A deep discussion on the communication system falls out the scope of the paper, but the availability of high band-width fiber optics channels suggests that physical high-speed broadcasting is going to become feasible.

The BAS model was successfully employed in different applications. The application to signal processing problems is reported in (ALDERIGHI et al, 1997 and 1997a). An example of application involving pure symbolic processing is shown in (ALDERIGHI et al, 1997b). These encouraging results lead us to think the BAS could be a promising approach to a wide category of design problems.

This paper describes our attempt to provide a suitable formalism for

the BAS in order to be able to analyse its properties, as relevant to correctness. Specifically it is shown how a correctness proof of the computation accomplished can be developed. In Section 2 the main features of the BAS are briefly reviewed, while the formalism is presented in Section 3. An example concerning the recognition of well-formed formulas in propositional calculus and the correctness proof are given in Sections 4 and 5.

2. Basic Concepts

The BAS models a collection of identical reactive entities (BAs) evolving concurrently and synchronously in response to a sequence of external stimuli generated by its environment and broadcast to all of them by means of a communication system.

Concurrence is meant as a co-operation between BAs for the accomplishment of a given computational task, that is the behaviour of a BA is affected by the BAs that are logically related to it (this concept will be clear in the sequel). In general this behaviour is also affected by the input data coming from the external environment.

BAs co-operation is based on the ability of each BA to observe other BAs. In the more general case, this visibility is global, that is information is broadcast to all BAs in the system.

The synchroniser unit provides the system timing, ensuring the simultaneous activation of all BAs, while control and computing activities of the entire system are spread over BAs.

The injector automaton represents the BAS interface to the external environment; it distributes the information received from the environment to all BAs. Data injection can be performed according to different modalities and protocols, depending on the specific application requirements.

2.1. Static Description

Many of the concepts and definitions that will be presented hereon are not novel and can be found in most literature concerning modelling of reactive systems, distribute systems and so on. What we are trying is to re-formulate these concepts in the context of systems based on a certain number of BAs. Where necessary we will indicate explicitly the differences of the BAS and cellular automata.

Generally speaking, a BA is a state-based entity capable of performing actions when some events trigger a transition of its state. There are external and internal events. External events are data coming from the external environment; internal events are the observable effects of the behaviour of the other BAs in the system. A BA has local memory and is able to perform

basic operations (arithmetical, logical, etc.). The state of a BA contains information about control and computing activities, conditions, data-items and related BAs. More specifically, three different parts can be identified: control, data and affinity.

The control part is relevant to BA's behaviour and identifies four functioning phases. In the first phase, the reset phase, BA is initialised; then, in the second phase data from the external environment are sensed through the injector; in the third phase, the activity phase, the BA performs the required computation concurrently with other active BAs; finally, in the fourth phase either a correct termination of the computation or some exception conditions are reached.

The data part contains information about the data-items used to perform the computation, namely values and their availability. The affinity part contains information about the affinity links of the BA.

The concept of affinity is fundamental for understanding the BAS model and how it differs from cellular automata. BAs evolve according to a set of state transition rules which are common to all BAs. State transition of each BA is based, like in cellular automata, on a certain number of BAs, but which BAs are involved is determined in a dynamic way, far from being set a priori on a topological basis. It is, thus, necessary that each BA knows the state of all the other BAs, though only a small part of this information may be necessary for state transition. The specialised dependence of the state transition function on the state of other BAs defines a relationship of state sensitivity among BAs, that we have defined as elective affinity. In the time step in which a BA becomes active its state transition is based on its own state and is affected by the state of any other BA: it is therefore affined to all BAs, or (which is the same) to none at all. As affinity links are established state sensitivity is more and more specialised, and is in the end restricted to only a few BAs.

To better understand this mechanism, let us consider the very simple case of a BAs class, the representatives of which are given the possibility to establish only one single affinity link, that is a BA be affined with just another one. Let $P(i, j)$ indicate a predicate over the states of generic BAs i and j. This predicate can act as a trigger for the establishment of affinity links: if BA i finds $P(i, j)$ to be true for a particular j, the i will be affined to j starting from the time step next to the observation. This means, in practice, that in a particular component of the state of BA i a part (a component or more) of BA j is stored. This part represents a sort of 'identification badge', by which BA i will recognise j from the others. When i observes the others, it will be interested only in the BA with this badge, that is j, in most cases.

In this special case the identification badge must be a unique label in order to grant uniqueness of affined BAs. This type of affinity link may be called 'point-to-point', as it corresponds to a specialised communication of BA j to BA i. More interesting cases of affinity may be established; first, by

simply dropping the request of uniqueness for the badge, an arbitrary number of BAs can be affined to each other, sharing the same badge. This kind of affinity link may be called 'open multicast'. Affinities may superimpose and cross, as far as they are related to different aspects of computation.

2.2. *Dynamic Description*

The behaviour of the BAS in correspondence with a given sequence of external data consists of a series of snapshots of the system's situation; such a snapshot is called *global state*. The first in the sequence is the initial global state, and each subsequent one is obtained from its predecessor by executing a step. This global state actually gathers the situations (state) of all BAs present in the system.

The functioning of the BAS is determined by the functioning of all BAs. At each step all BAs undergo three phases:

- *Prologue*: each BA observes the state of other BAs. This is the only way for BAs to exchange information, and can be done in the prologue only; the subsequent operation works on a consistent snapshot of the system state. As the state information is received through the communication system (the minimum requisite of which is the gossiping capability), it is stored in a local memory area.
- *Behaviour*: the current state, the observed state of other BAs, and the external datum (if any) trigger the state transition. As a result, the BA moves to a new state. Values of data-items and affinities may be modified.
- *Epilogue*: the new state is assigned and made visible, that is it is sent to the communication system.

The sequence of state transitions is meant to be loop-free and converge to one of the possible final states in a prefixed number of time steps. To this end some attention must be paid to rule formulation and state definition in order to avoid loops and also to grant deterministic behaviour. The general form of a state transition is $c(e) \rightarrow a$ a where e is (are) the event(s) that triggers the transition; c is a condition that guards the transition from being taken unless it is true when e occurs; a is the action that is performed when the transition is taken.

Fundamental for understanding the BAS is also the concept of stability. Indeed, it serves to characterise the correctness property of the system. In analogy with dynamic systems, the evolution of the entire system can be viewed as passing from a highly unstable state to more stable ones, until a highly stable state, the final state, is reached. This state may correspond to a legal termination of the computation (the highest stability) or to an anomalous functioning condition.

DEFINITION A BAS is said to be in a *stable* state (or similarly to be *stable*), if and only if all BAs are in a *stable* state (or similarly are *stable*).

DEFINITION A BA is said to be in a *stable* state (or similarly to be *stable*), if there are no external stimuli from the environment and no transition rules can be applied.

DEFINITION A BA is said to be in an *unstable* state (or similarly to be *unstable*), if transitions occur from this state, that are unconditional to the external stimuli from the environment.

DEFINITION A BAS is said to be in an *unstable* condition (or similarly to be *unstable*), if at least one BA is in an *unstable* state (or similarly is *unstable*).

There are two kinds of stable conditions: those corresponding to final states and those corresponding to error conditions. Let us define the former as *successfully stable* conditions, while the latter as unsuccessful ones.

DEFINITION A BAS is said to be in a *successful condition* (more briefly, it is successful) if all BAs are in a successfully stable state, or they are in the reset state.

DEFINITION A BAS is said to be in an *unsuccessful condition* (more briefly, it is unsuccessful) if at least one BA is not in a successfully stable state, nor it is in the reset state.

DEFINITION A BAS system is said to be *correct* if and only if 1) for every legal sequence of external stimuli the system is successful and 2) for any illegal sequence of external stimuli some BA does not reach a successful stability condition, nor it has left the reset state, that is the system is unsuccessful.

The assumptions we have adopted in defining the BAS are the following:

1. A BA in an unstable state will always reach a stable state.
2. Changes occurring in a BA during a step can be sensed only after the completion of the step (in the prologue phase).
3. Changes occurring locally to a BA cannot be observed.
4. At each step BA's behaviour is determined on the basis of the situation at the beginning of the step.
5. At each step, at most one transition per BA is taken.

3. Formalism

We formally denote a generic BAS consisting of N BAs by a $(N+1)$-uple of finite state BAs $(M_1, M_2, \ldots, M_N, M_{\text{INJ}})$, where $M_1, \ldots M_N, M_{\text{INJ}}$ generally denote Moore finite state BAs (HOPCROFT, 1979). An external synchroniser exists that provides the correct timing of their operation. M_is are identical and can be distinguished from each other.

A BA is described by a 7-uple $(Q, \Sigma, \Delta, D, \lambda, q_0, F)$. With respect to the definition in (HOPCROFT, 1979), the function D (behaviour Description) is introduced, replacing the classical state transition function δ. Indeed states are deeply structured, differently from the standard monolithic view, and this structuring affects BA's behaviour. A control is also defined for the BA that applies D in order to produce the behaviour.

$Q = Q_{\text{NAME}} \times Q_{\text{MAC}} \times Q_{\text{AFF}} \times Q_{\text{DATA}}$ is the set of states consisting of different components, called state components, defining respectively the BA's identifier, the set of control macrostates, the set of affinity-related microstates, and the set of data-related microstates. Q_{NAME} is the set of all possible identifiers and allows to distinguish BAs from each other. Q_{MAC}, as mentioned in Section 2, denotes control states identifying the main phases of BA's behaviour. $Q_{\text{AFF}} = Q_{\text{MIC\_AFF}} \times Q_{\text{VAL\_AFF}}$ describes affinity related information and is given by $Q_{\text{VAL\_AFF}} = V_1^{(a)} \times V_2^{(a)} \times \ldots \times V_{K(a)}^{(a)}$, whose components specify the affinity badges (point to point or open multicast) and $Q_{\text{MIC\_AFF}} = M_1^{(a)} \times M_2^{(a)} \times \ldots \times M_{K(a)}^{(a)}$, which is the set of corresponding control microstates. It is worth reminding that each $Q_{\text{VAL\_AFF}}$ component do not identifies a single affined BA, as the same badge can be held by different BAs at the same time. $Q_{\text{DATA}} = Q_{\text{MIC\_DATA}} \times Q_{\text{VAL\_DATA}}$ is used to describe transformations of data and is given by $Q_{\text{VAL\_DATA}} = V_1^{(d)} \times V_2^{(d)} \times \ldots \times V_{K(d)}^{(d)}$, whose components specify typed data, and $Q_{\text{MIC\_DATA}} = M_1^{(d)} \times M_2^{(d)} \times \ldots \times M_{K(d)}^{(d)}$ describing their related control microstates. Let us define Q_{CTRL} to be the set $Q_{\text{MAC}} \times Q_{\text{MIC\_DATA}} \times Q_{\text{MIC\_AFF}}$ and let us call its components *pure control* components.

$\Sigma = \left(\underbrace{\Delta \times \Delta \times \ldots \times \Delta}_{N\text{times}} \right) \times \Delta_{\text{INJ}}$ is the input alphabet, consisting of the Injector's output alphabet plus the output alphabets of all BAs. Except the input from the external environment, this is similar to what is defined in cellular BAs (GARZON, 1995) but in our case all BAs are interconnected.

Δ is the output alphabet.

D is the behaviour description function, which *de facto* corresponds to a state transition function: $D : \Sigma \times Q \to Q$. The description is accomplished by means of rules. Rules directly imply the BA's state transition diagram; this is associated to the pure control component Q_{CTRL}. Let us call this

diagram STD (State Transition Diagram). Alternatively, given an STD, rules can be associated thereto in straightforward way by specifying for each rule (1) which pair of nodes it refers to (they must be in an *antecedent-consequent* direct relationship) and (2) which pre- and post-conditions hold for the components other than Q_{CTRL} involved in the transition.

$\lambda : Q \rightarrow \Delta$ is the output function. It is a bi-jection between the state and output states.

q_0 is the initial, or reset, state. It contains prefixed values for Q_{MAC}, Q_{DATA} and Q_{AFF} that are common to all BAs, while for the component Q_{NAME} the value is univocally defined for each BA in order to distinguish BAs from each other.

F is the set of final states. If the BA enters one of these states it does not move to any next state (that is no transition rule is taken) as long as no external inputs are sensed. Yet transition may occur in presence of an external input in D_{INJ}.

Actually, the entire system can be viewed as a Moore state-based machine defined by the 7-uple $(Q_{\mathrm{SYS}}, \Sigma_{\mathrm{SYS}}, \Delta_{\mathrm{SYS}}, D_{\mathrm{SYS}}, \lambda_{\mathrm{SYS}}, q_{0\mathrm{SYS}}, F_{\mathrm{SYS}})$.

$Q_{\mathrm{SYS}} \left(\underbrace{Q \times Q \times \ldots \times Q}_{N\text{times}} \right) \times Q_{\mathrm{INJ}}$ is the set of states of all BAs (or global state) and is built starting from the BAs state sets, which are identical.

$\Sigma_{\mathrm{SYS}} = \left(\underbrace{\Delta \times \Delta \times \ldots \times \Delta}_{N\text{times}} \right) \times \Delta_{\mathrm{INJ}} \times \Sigma_{\mathrm{ENV}}$ is the input alphabet. Also Δ sets are identical.

$D_{\mathrm{SYS}} = \left(\underbrace{\Delta \times \Delta \times \ldots \times \Delta}_{N\text{times}} \right) \times \Delta_{\mathrm{INJ}}$ is the output alphabet.

D_{SYS} is the global transition function, or global dynamics according to (GARZON, 1995), defined as the $(N+1)$-tuple $((D_1, D_2, \ldots, D_N), D_{\mathrm{INJ}})$, containing the transition functions of all BAs, whose domains and ranges are contained in $(\Sigma_{SYS} \times Q_{\mathrm{SYS}})$ and Q_{SYS} respectively.

λ is the output function. It is a bi-jection between the global state set and the output alphabet Δ_{SYS}.

$q_{0\mathrm{SYS}}$ is the initial (or reset) state. It contains the initial state of all BAs.

F_{SYS} is the set of final states. Instead of a listing of final states criteria will be provided specifying which configurations of the system are to be considered as final.

Let us see more closely how to define each state transition rule. According to BA's observing activity during the prologue phase, let us name *observer* the current BA and *observed* the remaining BAs, objects of the actual observation. As mentioned in Section 2, each rule contains a guard,

i.e. a predicate, that has to be satisfied for the rule to be applied. It also contains a predicate expressing the condition that is to be held after taking the action corresponding to that transition. The action corresponds to an assignment of new values to some state components. The guarded predicate constitutes the rule pre-condition and is generally expressed as conjunction (&-products) of possible disjunctions (υ-sums) of elementary (equality) predicates over single state components. The post-condition is expressed in the same way as conjunction of equalities over state components. The language we have chosen for specifying transition rules is that of first-order logic, naturally well suited to express Boolean conditions.

The pre-condition usually involves the state of both observer and observed BAs. During observation, the states of all BAs in the system are sensed by the observer. Possible optimisations can imply, for instance, that only BAs that moved to a new state send this information, or, once that affinity relationships are established, the observer senses its affined automata only. The observer is not aware of the identity of observed BAs it senses; in other words, it cannot distinguish them from each other. This is to be reflected in transition rules for which observed-related information is to be different yet indistinguishable. A possible way to achieve this is to associate an index (local to each BA) to the state information of observed BAs, thus granting difference, and to require transition rules to be invariant with respect to these indexes. Invariance can be obtained by means of the following.

1. Rules that for their very nature are invariant. This is the case when universal or existential quantifiers are used in defining the rule;
2. Rules identical up to the index. This is the case when the index is a free variable of the rule. If there are N BAs in the system, then a same rule will be repeated N times.

Now we can better define some concepts related to stability of states and affinity, that will be useful in the correctness proof.

DEFINITION The STD of a BAS BA is a finite tree, called State Transition Tree (STT).

This derives from the requisite that transitions from unstable to stable states are loop-free state and uniquely defined. STT nodes are supposed to be identified uniquely by pure control state components. In the following we will therefore talk in the same way about nodes of the tree and control states of the BAs.

DEFINITION A thin sub-tree of an STT, whose root and leaf are respectively root and leaf of the STT, is called *history* of the STT. A history is uniquely determined by its leaf. Leaf depth in the tree defines the *history length.*

Actually, different histories correspond to a single STT, corresponding to the possible different root to leaf paths of the tree.

DEFINITION Given a history H, a state component is *k-stable in H* iff its value remains unchanged in all nodes of H of depth greater than k.

DEFINITION At a given step in the system evolution, a state component of a BA is *steady* iff:

1. the state component is k-stable in a history H;
2. the actual state of the BA is a node of H and its depth is greater than k.

State components which are non-trivially k-stable in a history can be used for establishing affinity links.

DEFINITION A state component is an *affinity badge* if it is a steady k-stable (k is smaller than the history length) for some observed BAs.

DEFINITION BA a is said to be *affined* to BA b iff some steady component of b is assigned the affinity badge of a. Affinity links define a digraph. Let us call it the *affinity graph.*

DEFINITION : A transition rule is called *affinity checking rule* iff its pre-condition contains an equality predicate (called *affinity checking predicate*) involving both a state component of the observer BA and an affinity badge of some observed BAs.

DEFINITION A rule is an *affinity establishing rule* iff its post-condition contains the assignment of the affinity badge value of some observed BAs to a state component of the observer BA.

DEFINITION A state component is an *affinity pointer* iff it is a k-stable component stable (k is smaller than the history length), it is assigned the value of an affinity badge by means of some affinity establishing rule, and any other occurrence of the component is in the pre-condition of affinity checking rules.

4. The Example

As an example we present a BAS system which accepts first order propositional logic formulae, according to a specific grammar, given below, and builds the parse trees of the accepted words with a top-down strategy. The system is required to reach a successful condition for all and only the words

of the language, while it must terminate in an unsuccessful state otherwise. During word acquisition the parse tree is constructed in a distributed manner and is constituted at the end by a certain number of stable BAs. Therefore the system is to be viewed more like a memory that checks writing attempts (it is strongly typed) than a simple(pure) language checker.

In the following we will show how a correctness proof in this case study can be carried out; for the sake of brevity we shall not deal with the parse tree construction process, but with the language checking problem.

The system is constituted, as said before, by an injector and a fixed number of BAs. The injector receives a string of symbols from the environment and injects them one at a time, according to the synchronisation rules. The total number of BAs is related with the maximum complexity of the formulae which can be correctly recognised, as each node of the parse tree is assigned to a single BA. This means that the acquisition of a formula can terminate in an unsuccessful condition even if the formula is correct. A precise definition of the accepted language which takes into account this limitation is given in the following.

4.1. Language Description

According to (GENTZEN, 1935) the language of first order propositional logic can be inductively constructed. A symbol which stands for an arbitrary proposition is a *propositional variable.* In the following the term *atom* will be used. An atom is a formula. If A is a formula then $!A$ is a formula (negation). If A and B are formulae then $A \,\&\, B$, $A \vee B$, $A \supset B$ are formulae too. Parentheses are used to avoid the definition of priority rules. Following these definitions an arbitrary well formed formula can be constructed. Formulae that may have arisen in the course of the construction, including the formula itself, are called sub-formulae.

In order to distinguish the formula itself from proper sub-formulae we refer to the concept of *root node* of a tree. The root of a formula plays a very important role in the correctness proof, exposed in the next paragraph. The root node of the parse tree of a formula is called *root of the formula.* If a formula is constituted only by an atom, then the root coincides with the only node of the parse tree.

A set of production rules which summarise all these concepts have been formulated .

⟨ atom root ⟩ → $A\#$ | $B\#$ | $C\#$

where "#" is the string terminator

⟨ formula root ⟩ → (⟨ atom ⟩⟨ connective ⟩⟨ atom >)# | (⟨ atom ⟩
⟨ connective ⟩⟨ proposition ⟩)# |
| (⟨ proposition ⟩⟨ connective ⟩⟨ atom ⟩)# | (⟨ proposition ⟩
⟨ connective ⟩⟨ proposition ⟩)# |

| (!⟨ proposition ⟩)# | (!⟨ atom ⟩)#
⟨ proposition ⟩ →| (< atom ⟩⟨ connective ⟩⟨ atom ⟩) | (< atom ⟩
⟨ connective⟩⟨ proposition ⟩) |
| (⟨ proposition ⟩⟨ connective⟩⟨ atom ⟩) | (⟨ proposition ⟩
⟨ connective⟩⟨ proposition ⟩) |
| (!⟨ proposition ⟩) | (!⟨ atom ⟩)
⟨ atom ⟩ → A | B | C
⟨ connective ⟩ → & | v |⟩

For the sake of clarity we define the alphabet of the language S = { "(", ")", "A", "B", "C", "!", "&", "v", "⊃", "#" }. The context-free language defined in this way is not the real language the system accepts. Indeed the system as a whole is a finite state BA and is obviously restricted to regular grammars. This limitation is de facto true for all finite systems (which covers the totality of real systems); we only want to remark it for our system.

A solution to this is the following language construction, which uses the application of a finite number of production rules. We use the formalism of regular expression (HOPCROFT, 1979): A = "A" + "B" + "C", $F_{!A} = (!A)$, c = "&" + "v" + "⊃", $F_{AcA} = (AcA)$.

Let us define the language P_k of propositions of fixed degree k.

$$
\begin{aligned}
P_0 &= A\,, \\
P_1 &= F_{!A} + F_{AcA}\,, \\
P_2 &= (AcF_{!A}) + (F_{!A}cA) + (F_{!A}cF_{!A}) + (F_{!A}cF_{AcA}) + \\
&\quad +(F_{AcA}cF_{!A}) + (F_{AcA}cF_{AcA}) + (AcF_{AcA}) + (F_{AcA}cA) + \\
&\quad +(!F_{!A}) + (!F_{AcA})\,.
\end{aligned}
$$

Elements in P_k can be indexed with integers in $\{1, \ldots, n_k\}$:

$$
\begin{aligned}
P_k &= \sum_{l=1}^{n_k} u_l^k\,, \\
P_{k+1} &= \sum_{\substack{i=0..k \\ l_1=1..n_k \\ l_2=1\cdot n_i}} (u_{l_1}^k c u_{l_2}^i) + \sum_{\substack{i=0..k \\ l_1=1..n_i \\ l_2=1\cdot n_k}} (u_{l_1}^i c u_{l_2}^k) + \sum_{l=1\cdot n_k} (!u_l^k)\,.
\end{aligned}
$$

We can now give the expression for the language L_k of the formulae with degree at most k

$$
L_k = \sum_{l=0}^{k} P_l\,.
$$

4.2. BA State Vector: Notation and Structure

Here we give some remarks about rule notation and the description of the state vector structure of a BA of the system, referring to the general outline given in Sections 2 and 3. As far as rule notation is concerned, we indicate with State.xxx the state components of observer BA, and adding a subscript $State_j$.xxx for observed BAs state. Examples of defined production rules are given in the Appendix. State parts and sub-parts, down to components, are denoted as dotted paths, in a C-like style. For example .mic.rgt denotes component *rgt* in the microstate part.

State vector is composed by four parts : *name*, that coincides with Q_{NAME}; *mac* macrostate, that coincides with Q_{MAC}; *mic* microstate, that contains both $Q_{\text{MIC\_AFF}}$ and $Q_{\text{MIC\_DATA}}$; *val*, a value that contains both $Q_{\text{VAL\_AFF}}$ and $Q_{\text{VAL\_DATA}}$.

Macrostate possible values have been labelled: **INA**ctive, **REA**dy, **F**ormula **R**oot **BA**, **F**ormula **BA**, **A**tom **R**oot **BA**, **A**tom **BA**, **EX_REA**dy, **ERR**or.

REA corresponds to the phase of sensitivity to injected stimuli; EX_REA macrostate is related to the successful state of a BA with REA macrostate, once the termination symbol is injected. FRA, FAU, ARA, AAU macrostates correspond to the activity phase and are related to the specific role the BA plays during the acquisition. Once a BA is assigned one of these values, it will not change the value, unless an exception condition is verified and a transition to the error macrostate occurs.

Microstate and value parts are sub-divided into five components each:

- *id*: identity ; components of $Q_{\text{MIC\_DATA}}$ and $Q_{\text{VAL\_DATA}}$
- *lft* : left ; components of $Q_{\text{MIC\_AFF}}$ and $Q_{\text{VAL\_AFF}}$
- *rgt*: right ; components of $Q_{\text{MIC\_AFF}}$ and $Q_{\text{VAL\_AFF}}$
- *con*: connective ; components of $Q_{\text{MIC\_DATA}}$ and $Q_{\text{VAL\_DATA}}$
- *atom*: components of $Q_{\text{MIC\_DATA}}$ and $Q_{\text{VAL\_DATA}}$

Microstate legal values for each component are enumerated in the Appendix, while value components are described here.

Identity component is dynamically acquired and is a 2-stable component in any history of the system; it is employed as affinity badge. Left and right components are affinity pointers corresponding to the two possible sons of a parse tree node. Connective component contains principal (same degree) connective of the (sub-) formula the node is root of. These last three components are used only by BAs with FRA or FAU macrostate. Atom component is, on the contrary, used only by BAs with ARA or AAU macrostate, and it contains information of the particular propositional variable (atom symbol) the node is assigned.

There are basically two types of nodes: formula nodes and atom nodes. Formula nodes have sons, atom nodes have not (they are leaves). The sons

of a formula node stand for the operands of the principal logical connective of the formula.

4.3. Affinities in the Example

Affinities are introduced in quite natural way, according to the syntactic structure of the language.

On the base of the production rules given previously, the parse tree of sentences of the language has two possible forms, as shown in the *Fig. 1.*

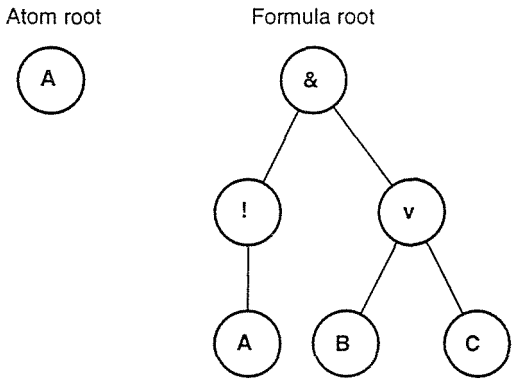

Fig. 1. Parse trees

The parse tree is constructed starting from the root in a depth-first fashion. The idea is to assign each node to a single BAS BA, which is in charge to verify the local consistency of tree construction. The edges of the tree are represented by references in the BAs state vector of (dynamical) names of other active BAs. The tree is represented in the system by the affinity digraph. Each active BA constitutes a node of the parse tree, and has to establish local links to possible son nodes. Moreover, its activity cannot end until the entire sub-tree has been built. A father formula node must therefore have some notion of his sons, in order to evaluate the state of completion of the sub-formula it is root of. This also implies that it cannot reach a success (end) state before all its sons do. All these concepts are implemented by making every formula node affined to its direct sons, with the additional request that a BA cannot reach a successful state before all affined BAs do.

The affinity establishing rules are A3.2, A4.2, A6.2, A7.2. State component val.id is used as affinity badge. In all affinity establishing rule preconditions it is checked whether the affinity badge of affine candidate is steady; this is realised easily, since val.id is a 2-stable component and it would be sufficient to verify that the macrostate is not INA. The affinity checking rules are A14.1, A15.1, A25.

4.4. Definition of Successful Condition

DEFINITION A single BA of a BAS is said to be *inactive* iff its macrostate is INA.

DEFINITION A single BA of a BAS is said to be *successful* iff its pure control components match with one of the patterns given in the following table.

Table 1. Termination successful conditions of a BA

Mac	Mic.lft	Mic.con	Mic.rgt	Mic.atom
FRA/FAU	NA	OK_M	OK_A	NA
FRA/FAU	NA	OK_M	OK_F	NA
FRA/FAU	OK_A	OK_D	OK_A	NA
FRA/FAU	OK_A	OK_D	OK_F	NA
FRA/FAU	OK_F	OK_D	OK_A	NA
FRA/FAU	OK_F	OK_D	OK_F	NA
ARA/AAU	NA	NA	NA	OK
EX_READY	NA	NA	NA	NA

DEFINITION A N automata BAS is in a successful condition (or, briefly, it is successful) iff all BAs are either successful or inactive.

DEFINITION A N automata BAS is in an unsuccessful condition iff it is not successful.

DEFINITION A N automata BAS is in an error condition iff at least one BA has ERR macrostate.

A consequence of the identicalness of all BAs in a BAS is that if a N automata BAS reaches a successful condition on a given input from the environment, then a $(N+K)$ automata BAS reaches a successful condition on the same input.

In the specific case, the definition of successful condition will be shown to be equivalent to:

DEFINITION A N automata BAS is in a successful condition iff there is one and only one BA with macrostate FRA or ARA which is successful.

DEFINITION A N automata BAS is said to accept language L if, for any word w in L, it reaches a condition in which there exists one and only BA in a successful state corresponding to the root of the parse tree of w, with FRA or ARA macrostate.

DEFINITION A N automata BAS is said to accept language L as a subformula if, for any word w in L, it reaches a condition in which there exists a BA in a successful state corresponding to the root of the parse tree of w, with FAU or AAU macrostate.

By a simple analysis of the possible histories of a BA and related transition rules it is therefore possible to formulate the following.

LEMMA *A BA with FRA or FAU macrostate cannot reach a successful state before its affined BAs do. Equivalently, if a BA with FRA or FAU macrostate has reached a successful state, then its affined BAs are also successful.*

5. The Correctness Proof

THEOREM *1 If $s \in L_k$ is injected, a N automata BAS, starting from reset state, reaches a successful condition.*

THEOREM *2 If a word s is injected into a N automata BAS and the system reaches a successful condition then $s \in L_k$.*

We will prove theorem 1 by induction over k. The basis is $k = 2$, and corresponds to theorem 1.1 .The inductive step is constituted by theorem 1.2. Theorem 2 will be proved by negation: if $s \notin L_k$ then the system reaches an error condition, and is not successful.

Therefore, we will prove that the system is correct, according to the definition given in Section 2.2.

THEOREM *1.1 If $s \in L_2$ is injected, a N automata BAS starts from reset state, if N is great enough, it accepts L_2.*

An exhaustive proof could easily be performed by means of a step by step simulation of a 7 automata BAS, on the basis of given rules. As remarked in the previous section, this is sufficient to prove it for a N automata BAS, with an arbitrary N. For the sake of brevity we do not report it here.

THEOREM *1.2 (Induction Step) If a N automata BAS, starting from reset state, accepts L_k, then, if N is great enough, it accepts L_{k+1}.*

In order to demonstrate this theorem we need to prove an additional lemma.

LEMMA *If a N automata BAS, starting from reset state, accepts L_k, then, if N is great enough, it accepts L_k as a subformula.*

Proof of Lemma

The demonstration in the case of atoms (elements of P_0) is implicit in the proof of theorem 1.2.

Let w be a non-atom element of L_k. Let us suppose that at a given time step BA a changes its macrostate from READY to FAU, becoming the formula BA corresponding to the root of w (a has READY macrostate when the first open parenthesis of w is injected). According to the rules, the evolution of the system is the same as in the case in which a has FRA macrostate, until the application of rules A13. Having a FAU macrostate, a applies one of rules A13.x instead of A13.xbis, changing its right microstate to CREATED_A or CREATED_F. In the next time step, according to rules A16.x right microstate of a changes to OK_A or OK_F, and the BA is therefore successful.

Proof of Theorem 1.2 Since $L_{k+1} = \sum_{l=0}^{k+1} P_l$, then is sufficient to demonstrate that the system accepts P_{k+1}. Let then a word in P_{k+1} be injected. Let us consider the evolution of the system, starting from reset state. Numbering indicates global time steps.

1. A BA, let us call it a, changes its macrostate from INACTIVE to READY (rule A1).
2. Symbol "(" is injected.
3. Possible rules for a are now A2 and A2bis. A2 is not applicable because there is no BA with FRA (Formula Root BA) macrostate. Hence, a-State.mac=FRA (Formula Root BA) and a-State.mic.sin=SCAN on the base of rule A2bis.
4. Another BA, let us call it b, changes its macrostate from INACTIVE to READY (rule A1).
   At time step 5 either "!", ⟨ atom ⟩ or "(" symbols can be injected. We shall include the demonstration for the first case only; the other two can be treated in a similar way. According to the language description, these cases cover all possible situations, and therefore theorem 1.2 is proved.
5. Symbol "!" is injected.
   After five time steps (we omit details here), the system reaches a condition in which: BA a is formula root, BA b is formula and is preparing to acquire a L_k word, and BA c has REA macrostate. BA b is affined to a, and hence is related to a son node of a.

Since by hypothesis the system accepts L_k , as a consequence of the lemma it accepts words in L_k as sub-formulae. This means that BA b, which is root of a word in L_k is proved to reach a successful condition. In particular, at a given time step t, it will change its right microstate to END_A or END_F (depending on right operand was an atom or a formula). After five time steps (we omit details here), the system reaches a condition in which: BA a is successful, BA b and m other BAs corresponding to the

parse tree b is root of are successful, and at most one additional BA has EX-READY macrostate, which is a successful state. The other BAs are inactive.

We have shown that if the system contains at least $m+3$ BAs it reaches a successful condition.

Proof. We prove that if a word $s \notin L_k$ is injected in a N BAs BAS system, it reaches an error condition.

We can assume that $s \in S^* - L_k$, where S^* indicates the Kleene closure of the alphabet S. Moreover, the last symbol of the word is always the terminator, since this property is granted by the injector.

On the base of the production rules given in Section 4.1 such string must verify at least one of the following conditions:

1) It starts with other symbols than "(" or "A" or "B" or "C"
2) It contains one of the following couples of symbols
   "(", one of "&", "v", "⊃", ")", "#"
3) It contains a sub-string of the form:
   "(!", one of "!", "&", "v", "⊃", ")", "#"
   "(⟨ proposition ⟩⟨ connective ⟩", one of "!", "&", "v", "⊃", ")", "#"
4) It contains a sub-string of the form:
   "(⟨ atom ⟩", one of "A", "B", "C", "!", "(", ")", "#"
   "(⟨ proposition ⟩", one of "A", "B", "C", "!", "(", ")", "#"
5) It contains a sub-string of the form:
   "(!⟨ atom ⟩", one of "A", "B", "C", "!", "&", "v", "⊃", "(", "#"
   "(⟨ atom ⟩⟨ connective ⟩⟨ atom ⟩", one of "A", "B", "C", "!", "&", "v", "⊃", "(", "#"
   "(⟨ atom ⟩⟨ connective ⟩⟨ proposition ⟩", one of "A", "B", "C", "!", "&", "v", "⊃", "(", "#"
   "(⟨ proposition ⟩⟨ connective ⟩⟨ atom ⟩", one of "A", "B", "C", "!", "&", "v", "⊃", "(", "#"
   "(⟨ proposition ⟩⟨ connective ⟩⟨ proposition ⟩", one of "A", "B", "C", "!", "&", "v", "⊃", "(", "#"
   "(⟨ proposition ⟩", one of "A", "B", "C", "!", "&", "v", "⊃", "(", "#"
6) It starts with:
   ⟨ atom ⟩, one of "A", "B", "C", "!", "&", "v", "⊃", "(", ")"
7) It starts with:
   ⟨ proposition ⟩, one of "A", "B", "C", "!", "&", "v", "⊃", "(", ")"

We shall prove that in each case the system contains at least a BA for which it is applicable a rule which changes BA macrostate to ERR. In all cases except case 1 this is accomplished by means of proofs similar to the demonstration of theorem 1, where it was necessary to demonstrate that the system contained one BA with certain state components. Case 1 is very

simple: the first BA in the system which acquires a REA macrostate verifies rule 18 pre-condition, which causes the BA to change its macrostate to ERR.

As far as the remaining cases are concerned, in *Table 2* we give here only the state values of interest and the rules which bring the system into an error condition.

Table 2. States in which syntactical errors are detected and related rules

Case	Mac	Mic.lft	Mic.con	Mic.rgt	Mic.atom	Rule
2	FRA/FAU	SCAN	NA	NA	NA	A19
3	FRA/FAU	OK_A/OK_F	OK_D	SCAN	NA	A20
	"	NA	OK_M	"	"	"
4	FRA/FAU	OK_A/OK_F	SCAN	NA	NA	A21
5	FRA/FAU	OK_A/OK_F	OK_D	END_A/END_F	NA	A22
	"	NA	OK_M	"	"	"
6	ARA	NA	NA	NA	OK	A23
7	FRA	OK_A/OK_F	OK_D	OK_A/OK_F	NA	A24
	"	NA	OK_M	"	"	"

6. Conclusions

The adoption of the BAS and the simple formalism introduced has allowed to verify the goodness of the solution to our language recognition problem. The correctness proof has been derived in a straigthforward way from the transition rules defined for the BAs and carried out without employing any automatic proof development system. The correctness proof has been done for a BAS consisting of arbitrary number of BAs.

Due to identicalness of BAs, in general it is always possible to prove properties about a system constituted by an arbitrary number of BAs, if the properties hold for the system with a lower number of BAs.

From the design point of view, the BAS approach is interesting because the description of the global system can be obtained starting from the description of a single BA. Indeed, system's behavioural specification is provided by the set of rules defined for a single BA. By means of specific requisites of stability dependencies among automata, the analysis of system behaviour can be accomplished by analysing the behaviour of a prefixed number of automata, playing a particular role in the system.

Finally, the BAS is well suited to model solutions that present strong inter-dependence relationships among data. Elective affinity is an effective way to achieve this. Relationships among data are naturally mapped into relationships among automata.

References

[1] ALDERIGHI, M. – MAZZEI, R. P. G. – SECHI, G. R. – TISATO, F. (1997): Broadcast Automata: a Parallel Scalable Architecture for Prototypal Embedded Processors for Space Applications, *Proc. of the Thirtieth Annual Hawai'i International Conference on System Sciences*, (Maui, Hawaii, January 7–10), Vol. V, IEEE Press, pp. 208–217.

[2] ALDERIGHI, M. – MAZZEI, R. P. G. – SALA, A. – SECHI, G. R. (1997a): Morphological Classification of CCD Frames in a Photon Counting Intensified CCD, *Proc. of the 1997 IEEE Instrumentation and Measurement Technology Conference (IMTC'97)*, (Ottawa, Canada, May 19–21, 1997), IEEE Press, pp. 118–123.

[3] ALDERIGHI, M. – BORDONI, A. – MOJOLI, G. – SALA, A. – SECHI, G. R. – VINATI, S. (1997b): Towards Models of Realistic Machines in Theoretical Computer Science, *2nd Workshop on Trends in Theoretical Informatics*, (Budapest, Hungary, March 9-14, 1997), in this volume.

[4] CHANDY, K. M. – MISRA, J. (1988): Parallel Program Design, A Foundation. Addison-Wesley, 1988.

[5] DEDERICHS, F. – DENDORFER, C. – WEBER, R. (1993): FOCUS: A Formal Design Method for Distributed Systems, LNCS 732, Springer, 1993.

[6] GARZON, M. (1995): Models of Massive Parallelism, EATCS, Springer, 1995.

[7] GENTZEN, G. (1935): Untersuchungen über das logische Schliessen, *Mathematische Zeitschrift*, Vol. 39, (1935), pp. 176–210, 405–431.

[8] HAREL, D. (1987): Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, Vol. 8, 1987, pp. 231–274.

[9] HOARE, C. A. R. (1985): Communicating Sequential Processes, Prentice-Hall, 1985.

[10] JONSSON, B. (1989): A Fully Abstract Trace Model for Dataflow Networks, *Proc. 16th Annual ACM Symposium on Principles of Programming Languages*, 1989, pp. 155–165

[11] MILNER (1980): A Calculus of Communicating Systems, LNCS 92, Springer.

[12] REISIG (1985): Petri Nets. An Introduction, EATCS Monograph 4, Springer.

Appendix

mic.id = { Not Available | **OK** }

mic.lft = { Not Available | **SCAN** | **ID** | Wait | **NEST** | Wait for **NEST** | **OK_F** | **OK_A** }

mic.con = { Not Available | **SCAN** | **OK_M** | **OK_D** }

mic.rgt = { Not Available | **SCAN** | **ID** | **WAIT** | **NEST** | Wait for **NEST** | **END_A** | **END_F** | **CREATED_A** | **CREATED_F** | **OK_F** | **OK_A** }

mic.atom = { Not Available | **CREATED** | **OK** }

Only the most significant state transition rules are reported.

A1) INACTIVE → READY
(State.mac = INA) & (!∃j((State$_j$.mac = INA) & (State$_j$.name < State.name))) & !∃j(State$_j$.mac = REA) ⇒ (State.mac = REA) & (State.val.id = max$_j$
(State$_j$.val.id) + 1);

A2) Formula BA creation
(State.mac = REA) & (StateInjector.sym = "(") & ∃j(State$_j$.mac = FRA) & !∃j(State$_j$.mac = ARA) ⇒ (State.mac = FAU) & (State.mic.lft = SCAN);

A2bis) Formula root BA creation
(State.mac = REA) & (StateInjector.sym = "(") & !∃j(State$_j$.mac = FRA) & !∃j(State$_j$.mac = ARA) ⇒ (State.mac = FRA) & (State.mic.lft = SCAN);

A13) Formula termination
A13.1) (State.mac = FAU) & (State.mic.rgt = END_A)) & (StateInjector.sym = ")") ⇒ (State.mic.rgt = CREATED_A);
A13.1bis) (State.mac = FRA) & (State.mic.rgt = END_A)) & (StateInjector.sym = ")") ⇒ (State.mic.rgt = OK_A);
A13.2) (State.mac = FAU) & (State.mic.rgt = END_F) ⇒ (State.mic.rgt = CREATED_F);
A13.2bis) (State.mac = FRA)) & (State.mic.rgt = END_F) ⇒ (State.mic.rgt = OK_F);

A16) Formula final transition
A16.1) ((State.mac = FAU) v (State.mac = FRA)) & (State.mic.rgt = CREATED_A) ⇒ (State.mic.rgt = OK_A);
A16.2) ((State.mac = FAU) v (State.mac = FRA)) & (State.mic.rgt = CREATED_F) ⇒ (State.mic.rgt = OK_F);

A18) First symbol is unacceptable
(State.mac = REA) & (State.mic.lft = SCAN) & (StateInjector.sym = { "!" | "&" | "v" | ">" | ")" | "#" }) & ∃j(State$_j$.mac = FRA) & !∃j(State$_j$.mac = ARA) ⇒ (State.mac = ERR)

A19) Left symbol is unacceptable
((State.mac = FAU) v (State.mac = FRA)) & (State.mic.lft = SCAN) & (StateInjector.sym = { "&" | "v" | ">" | ")" | "#" }) ⇒ (State.mac = ERR)

A20) Right symbol is unacceptable
((State.mac = FAU) v (State.mac = FRA)) & (State.mic.rgt = SCAN) & (StateInjector.sym = { "!" | "&" | "v" | ">" | ")" | "#" }) ⇒ (State.mac= ERR)

A21) Connective symbol is unacceptable
((State.mac = FAU) v (State.mac = FRA)) & (State.mic.con = SCAN) & (StateInjector.sym = { "A" | "B" | "C" | "!" | "(" | ")" | "#" }) ⇒ (State.mac = ERR)

A22) Formula end symbol is unacceptable
((State.mac = FAU) v (State.mac = FRA)) & ((State.mic.rgt = END_A) v (State.mic.rgt = END_F)) & (StateInjector.sym = { "A" | "B" | "C" | "!" | "&" | "v" | ">" | "(" | "#" }) ⇒ (State.mac = ERR)

A23) Input after atom root end
(State.mac = ARA) & (State.mic.atom = OK) & (StateInjector.sym = { "A" | "B" | "C" | "!" | "&" | "v" | ">" | "(" | ")" }) ⇒ (State.mac = ERR)

A24) Input after formula root end (State.mac = FRA) & ((State.mic.rgt = OK_A) v (State.mic.rgt = OK_F)) & (StateInjector.sym = { "A" | "B" | "C" | "!" | "&" | "v" | ">" | "(" | ")" }) ⇒ (State.mac = ERR)