Interacting Automata for Modelling Distributed Systems^{*}

Irina A. Lomazova

Program Systems Institute of the Russian Academy of Science
Pereslavl-Zalessky, 152020, Russia
`irina@univ.botik.ru`

Abstract. A *community of interacting automata* is a set of nondeterministic finite automata which can execute actions autonomously, synchronize with each other, or generate new members of the community. Thus interacting automata allow to model distributed systems with unlimited number of interacting agents. We show that the formalism of interacting automata has a clear semantics and nice semantic properties, leading to decidability of some important behavioral properties of modelled systems.

1 Introduction

Finite automata are widely used in modelling systems, especially when handling control aspects is important. The benefits of finite automata are in particular connected with clear mathematical semantics, systems visualization and decidability of many crucial behavioral properties.

The up-to-date needs for modelling concurrent and distributed systems with a complex structure lead to different extensions of the finite automata formalism, such as hierarchical, interacting, dynamic automata [1,5,9,10], which allow to model hierarchy, concurrency and communication among concurrent components.

Here we introduce the *interacting automata*, which together with concurrency and communication allow to model dynamic activation of new agents. There is a rather large variety of formalisms for representing distributed systems in the modern theoretical computer science. Modelling systems with a dynamically changing number of agents requires extending standard formalisms to capture a possibility of changing a system structure (see e.g. [6] for an extension of Petri nets for modelling multi-agent systems).

Interacting automata represent in some sense the most simple model of a system consisting of agents, which can do autonomous actions, interact with each other, be generated by other agents and die. Interacting agents form an unstructured community with no hierarchical structure, all agents have thus equal

^{*} This work was partly supported by the Presidium of the Russian Academy of Science, program “Intellectual computer systems”, project 2.3 – “Instrumental software for dynamic intellectual systems” and INTAS-RFBR (Grant 01-01-04003).

rights to do actions autonomously or synchronize. Agents are represented by finite automata. Initially there is one automata which can generate new members of community, which in turn can also generate new members. An agent automata dies, when it comes to its terminal state. Concurrent behavior is represented by interleaving semantics. A community comes to its final state, when it is empty, i.e. all its members have died.

The paper is organized as follows. In section 2 we give the definitions of interacting automata community. In section 3 we study languages, accepted by interacting automata. It is shown, that interacting automata accept some context-free nonregular and even some noncontext-free languages. In section 4 it is shown, that interacting automata can be considered as well-structured transition systems, and this leads to decidability of some behavioral properties for them. In section 5 interacting automata are compared with Petri nets. It is proved, that interacting automata can be simulated by Petri nets. As the consequence of this result we obtain decidability of the Reachability problem for interacting automata. Also it implies, that there exist a context-free language, which is not accepted by interacting automata. Together with the results of section 3 this means, that the class of interacting automata languages is incomparable with the class of context-free languages.

2 Interacting Automata

We start by recalling that a (nondeterministic) finite automaton is defined as a tuple $A = (Q, U, \theta, q_0, F)$, where

- Q is a finite set of inner states;
- F — a finite set of terminal states, $F \cap Q = \emptyset$;
- U — a finite alphabet of action names;
- $\theta : Q \times U \rightarrow 2^{(Q \cup F)}$ — a transition map;
- $q_0 \in Q$ — an initial state.

Here the set F of terminal states is not included into the set Q of inner states (as it is usually done in the standard definition of a finite automaton) by reasons of convenience. In our definition the transition map can't be applied to terminal states, so after coming to any terminal state an automaton stops. It can be easily shown, that our definition is equivalent to the standard one concerning expressibility, accepted languages and other behavioral properties of automata. We do not consider here the questions of automata size and minimization.

A finite automaton can be considered as a finite oriented graph with a set $Q \cup F$ of nodes and with arcs labelled by symbols from U . As usual, we suppose U contains a special symbol τ for the silent (invisible) action. In the graph representation inner states are designated by circles. The set Q of inner states contains a designated initial state q_0 . Terminal nodes are designated by squares and do not have outgoing arcs.

A community of interacting automata will be a multiset of finite automata with actions of three types: usual autonomous actions, actions of synchronization, and actions generating a finite automaton — a new member of the community.

Recall, that a *multiset* m over a set S is a mapping $m : S \rightarrow \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, i.e. a multiset may contain several copies of the same element. A multiset m is *finite* iff the set $\{s \in S \mid m(s) > 0\}$ is finite. For two multisets m, m' we write $m \subseteq m'$ iff $\forall s \in S : m(s) \leq m'(s)$ (the inclusion relation). The sum of two multisets m and m' is defined as usual: $\forall s \in S : (m + m')(s) = m(s) + m'(s)$. We define also $(m - m')(s) = m(s) - m'(s)$, if $m(s) > m'(s)$, and $(m - m')(s) = 0$, otherwise.

Definition 2.1. Let L be a finite set of synchronizing labels, s.t. for each label $l \in L$ the adjacent label $\bar{l} \in L$ is defined and $\bar{\bar{l}} = l$. Let U be a finite set of action names, $\mathcal{A} = \{\alpha_1, \dots, \alpha_k\}$ — a set of automata names. A community of interacting automata is defined as a collection $\Sigma = (A_1, \dots, A_k)$ ($k \geq 1$) of finite nondeterministic automata, where for $i = 1, \dots, k$ an automaton $A_i = (\alpha_i, Q_i, (U \cup L \cup \mathcal{A}), \theta_i, q_0^i, F_i)$ is specified by the following components:

- α_i — a name of the automaton A_i ;
- Q_i — a finite set of inner states of the automaton A_i ;
- $(U \cup L \cup \mathcal{A})$ — a (common for all automata) set of names for autonomous (U), synchronized (L) and generating ($\mathcal{A}$) actions;
- $\theta_i : Q_i \times (U \cup L \cup \mathcal{A}) \rightarrow 2^{(Q_i \cup F_i)}$ — a transition map for A_i ;
- $q_0^i \in Q_i$ — an initial state for A_i , and
- F_i — a set of terminal states for A_i .

Among this collection a distinguished parent automaton is chosen, we suppose it to be A_1 .

Now we define the behavior of an automata community $\Sigma = (A_1, \dots, A_k)$.

A *configuration* of an automaton A_i is represented by a pair (α_i, q) , where α_i is the name, $q \in Q_i \cup F_i$ — a state of A_i . An automaton configuration (α_i, q) is called *terminal*, iff $q \in F_i$ — a terminal state.

A *configuration of an automata community* Σ is a multiset K over the set of all nonterminal configurations of automata $A_1, \dots, A_k$.

A *step* for an automata community Σ is a triple (K, u, K') , where K, K' — configurations for Σ , $u \in (U \cup \mathcal{A} \cup \{\tau\})$ — an action name. We define three kinds of steps.

- **An autonomous step.** Let the configuration K include an automaton configuration (α_i, q) and for some $u \in U$ we have $q' \in \theta_i(q, u)$, i.e. an autonomous action u transforms a state q of the automaton A_i into the state q' . Then the configuration K of Σ is transformed to K' (write $K \xrightarrow{u} K'$), where $K' = K - (\alpha_i, q) + (\alpha_i, q')$ provided $q' \in Q_i$ and $K' = K - (\alpha_i, q)$, if $q' \in F_i$. Thus terminal automaton configurations are deleted from the community.
- **A synchronization step.** Let K include two automata configurations (α_i, q) , (α_j, p) and for some label $l \in L$, $q' \in \theta_i(q, l)$, $p' \in \theta_j(p, \bar{l})$. Then K can be transformed to K' (write $K \xrightarrow{\tau} K'$), where K' is obtained by deleting all terminal automaton configurations from $K - (\alpha_i, q) - (\alpha_j, p) + (\alpha_i, q') + (\alpha_j, p')$.

- **A generating step.** Let K include an automaton configuration (α_i, q) and for some automaton name $\alpha_j \in \mathcal{A}$ we have $q' \in \theta_i(q, \alpha_j)$. Then K can be transformed to K' (write $K \xrightarrow{\alpha_j} K'$), where K' is obtained by deleting a terminal automaton configuration (if any) from $K - (\alpha_i, q) + (\alpha_i, q') + (\alpha_j, q_0^j)$.

We write $K \rightarrow K'$, if there exists a step from K to K' .

An automata community behavior is defined in terms of *consecutive runs* (or just *runs*). A run R is a finite or infinite sequence of automata community configurations $K_0, K_1, \dots$, such that

1. The *initial configuration* K_0 consists of the only automaton configuration (α_1, q_0^1) , i.e. the parent automaton in its initial state;
2. $K_i \rightarrow K_{i+1}$ for all $i = 0, 1, \dots$;
3. R is finite iff it can't be continued.

Thus the last configuration in a finite run may be either empty (a terminating run), or a nonempty configuration in which no action can occur (a deadlock). The *behavior* of an automata community Σ is defined as the set of all its runs.

3 Languages of Interacting Automata

Figure 1 shows an example of very simple automata community Σ_1 without synchronization; Σ_1 consists of the only automaton A , which can generate its own copies.

Fig. 1. The automata community Σ_1 .

Here a is the name of the automaton A , $A = (a, Q, (U \cup \mathcal{A}), \theta, q_0, F)$, where $Q = \{q_0, q_1, q_2\}$ is the set of inner states of A with the initial state q_0 , $U = \{b, \tau\}$ — the set of autonomous actions, the set of synchronization labels is empty, $\mathcal{A} = \{a\}$ — the set of automata names, θ — the transition map, shown by arcs in the graph representation of A , $F = \{q_3, q_4, q_5\}$ is the set of terminal states.

The automaton A can

- either do a silent action and terminate,
- or can generate its own copy, after that at some further moment do the action b and terminate,

- or first do the action b , then at some further moment generate its own copy and also terminate.

The visible actions of the automata community, generated by the automaton A , are actions a and b . During the life cycle of this automata community the unlimited number of copies of A can appear.

Definition 3.1. *Let Σ be an automata community. The language $\mathcal{L}(\Sigma)$ accepted by Σ is the set of all finite words over the alphabet $(U \cup \mathcal{A}) \setminus \{\tau\}$, s.t. for each word $w \in \mathcal{L}(\Sigma)$ there exists a terminating run R for Σ with a sequence w of visible actions.*

It's easy to see, that for an automata community Σ_1 , represented in Figure 1, $\mathcal{L}(\Sigma_1) = \{w \mid w \in \{a, b\}^* \text{ and } n_a(w) = n_b(w)\}$, where $n_a(w)$ and $n_b(w)$ designate the number of symbols 'a' (correspondingly, symbols 'b') in the word w .

Since the language $\mathcal{L}(\Sigma_1)$ can't be accepted by any finite nondeterministic automaton, the immediate consequence of this example is the following

Proposition 3.2. *The formalism of automata communities is strictly more expressive than nondeterministic finite automata.*

It is well-known, that the language $\mathcal{L}(\Sigma_1)$ is a context-free language, so it can be accepted by a push-down automaton. Further in Section 5 we show that there are context-free languages, that are not accepted by any automata community.

Fig. 2. The automata community Σ_2 .

However, interacting automata can accept also some languages, which are not context-free. Figure 2 shows the automata community Σ_2 , which accepts the language $\{a^n b^n c^n \mid n \geq 1\}$. It is well-known, that this language is not context-free and can't be accepted by any push-down automaton.

4 Interacting Automata as Well-Structured Transition Systems

Interacting automata may generate new members of their community, and hence they are infinite state-space systems. In this section we show, that interacting

automata satisfy a nice monotonicity property: a bigger community can produce all actions that are possible for a smaller one. Moreover, we prove that interacting automata can be considered as a Well-Structured Transition System (WSTS) [2]. The theory of WSTS helps to obtain some decidability results for interacting automata.

Recall that a transition system is a pair $\mathcal{S} = \langle S, \rightarrow \rangle$, where S is an abstract set of states (or configurations) and $\rightarrow \subseteq S \times S$ is any transition relation. For a transition system $\mathcal{S} = \langle S, \rightarrow \rangle$ we write $\text{Succ}(s)$ for the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of s . $\mathcal{S}$ is finitely branching if all $\text{Succ}(s)$ are finite.

Definition 4.1. A quasi-ordering is any reflexive and transitive relation $\leq$ (over some set X). A well-quasi-ordering (a wqo) is any quasi-ordering $\leq$ such that, for any infinite sequence $x_0, x_1, x_2, \dots$, in X , there exist indexes $i < j$ with $x_i \leq x_j$.

Note, that if $\leq$ is a wqo, then any infinite sequence contains an infinite increasing subsequence: $x_{i_0} \leq x_{i_1} \leq x_{i_2} \dots$.

Definition 4.2. A well-structured transition system (a WSTS) is a transition system $\Sigma = \langle S, \rightarrow, \leq \rangle$ equipped with an ordering $\leq$ between states such that $\leq$ is a wqo, and $\leq$ is “compatible” with $\rightarrow$, where “compatible” means that for all $s_1 \leq t_1$, and transition $s_1 \rightarrow s_2$, there exists a transition $t_1 \rightarrow t_2$, such that $s_2 \leq t_2$.

Now we define a wqo on configurations of an automata community and show that together with transformation relation $K \rightarrow K'$ on configurations it forms WSTS.

Theorem 4.3. Let Σ be an automata community, $\mathcal{K}$ — the set of all configurations for Σ . Then

- the relation $\subseteq$ of multiset inclusion on $\mathcal{K}$ is a wqo,
- if $K_1 \subseteq K_2$ are configurations for Σ and $K_1 \rightarrow K'_1$, then there exists a configuration K'_2 , s.t. $K_2 \rightarrow K'_2$ and $K'_1 \subseteq K'_2$.

Thus every automata community is a WSTS w.r.t. the multiset inclusion relation.

The following diagram illustrates the statement of the theorem:

$$\begin{array}{ccc} K_1 & \subseteq & K_2 \\ \downarrow u & & \downarrow u \\ K'_1 & \subseteq & K'_2 \end{array}$$

Proof sketch: To prove that the multiset inclusion on $\mathcal{K}$ is a wqo note, that for a given automata community the number of different automaton configurations is finite. So, configurations of an automata community are multisets over a finite set of automaton configurations and can be encoded by vectors of some fixed size n with nonnegative integers as elements. The multiset inclusion then corresponds to the component-wise comparison of integer vector elements, which is a wqo.

Compatibility is proved straightforward from the definition of an automata community behavior by the analysis of all three kinds of steps. $\square$

We say, that a configuration K accepts an action word ω , if there exists a sequence of steps $K \rightarrow K_1 \rightarrow \dots$ with the sequence ω of visible actions.

Corollary 4.4. *Let K, K' be two configurations for automata community Σ , such that $K \subseteq K'$. Then K accepts an action word ω implies K' accepts ω .*

Though a transition system with an infinite state-space has an infinite reachability tree, for each WSTS with an effective ordering relation and computable transition relation a finite coverability tree can be effectively constructed [3]. For interacting automata it can be done as follows.

We start from the root node labelled by the initial configuration K_0 . It is easy to see, that the number of configurations K , s.t. $K_0 \rightarrow K$, is finite. For each such K we construct a node, labelled by K and an arc from K_0 to K .

Then we continue this process taking each K instead of K_0 in turn, i.e. we start to construct the reachability tree. Each branch of this tree is continued until one of the following situations occurs:

- 1) we reach the terminal empty configuration, which is a leaf, or
- 2) we reach a deadlock configuration, which is also a leaf, or
- 3) we reach a configuration K , s.t. the branch leading to K contains a node, labelled by a configuration K' and $K' \subseteq K$, and then we also do not continue further this branch and declare the node K to be a leaf.

Since the multiset inclusion relation is a wqo, all branches in a coverability tree are finite. Since automata communities are finitely branching, for every automata community its coverability tree is finite. A coverability tree can be used for checking some behavioral properties of a modelled system.

A system terminates if there exists no infinite run (*Termination Problem*). The *Control-State Maintainability Problem* is to decide, given an initial state K and a finite set $\{K_1, K_2, \dots, K_m\}$ of configurations, whether there exists a computation starting from K with all its inner configurations covering (not less than w.r.t. the multiset inclusion) one of the K_i 's. The dual problem, called the *Inevitability Problem*, is to decide whether all computations starting from K eventually visit a state not covering one of the K_i 's. Thus for example, for automata communities we can ask whether there is a run with all configurations containing at least one copy of a given automaton (control-state maintainability), or whether all copies of a given automaton, which can evolve during a life-cycle of an automata community, will eventually come to its terminating state and disappear (inevitability).

Theorem 4.5. *Termination, the control-state maintainability problem and the inevitability problem (w.r.t. $\subseteq$) are decidable for automata communities.*

Proof sketch: An automata community doesn't have infinite runs iff all leaves in its coverability tree are marked by either empty configurations or deadlocks.

For a given automata community a computation starting from K with all its inner configurations not less than one of $\{K_1, K_2, \dots, K_m\}$ exists iff in the coverability tree with the configuration K in the root there exists a path from the root to a leaf with all inner configurations not less than one of the K_i 's. $\square$

5 Interacting Automata and Petri Nets

In this section we compare automata communities with Petri nets — a popular formalism for modelling and analysis of parallel and distributed systems [8].

Definition 5.1. *A Petri net is a finite directed graph with two types of nodes, referred to as places and transitions. An arc in a Petri net goes either from a place to transition, or from a transition to a place.*

For a transition t every place p with an arc going from p to t is called an input place, and a place q with an arc going from t to q is called an output place.

A marking M in a Petri net is a mapping from the set P of places to non-negative integers. We say, that there are k tokens in the place p , if $M(p) = k$. A marked Petri net is a Petri net together with its initial marking.

We now define the behavior of a Petri net. A transition t is enabled in a marking M iff each input place of t contains at M at least one token. An enabled transition t may fire by removing one token from every input (for t) place and by adding one token to every its output place.

A Petri net run is defined as a sequence of its firings starting from the initial marking.

Transitions in a Petri net may be labelled by names of actions to make a behavior of a Petri net observable. Some of transitions may be invisible (labelled by a silent action τ).

We show, that the behavior of every automata community can be simulated by a labelled Petri net. Let $\Sigma = (A_1, \dots, A_k)$ be an automata community with inner states $Q_1, \dots, Q_k$, terminal states $F_1, \dots, F_k$ and transition maps $\theta_1, \dots, \theta_k$ correspondingly. Without loss of generality we can suppose, that all sets of inner and terminal states here are pairwise disjoint. A Petri net $PN(\Sigma)$ simulating Σ will be constructed as follows:

1. For each inner state $q \in Q_1 \cup \dots \cup Q_k$ there will be a place $\tilde{q}$ in $PN(\Sigma)$. A configuration K for Σ will be represented by a marking $\tilde{K}$, so that n copies of an automaton configuration (α, q) will be encoded by n tokens residing in the place $\tilde{q}$.
2. For each arc (q, q') , labelled by an internal action u , in an automaton A_i of the community Σ we construct a transition t in $PN(\Sigma)$, also labelled by u , with the input place $\tilde{q}$ and the output place $\tilde{q}'$. If q' is a terminal state in A_i , then the transition t doesn't have output places.
3. For each arc (q, q') , labelled by the name α_i of an automaton A_i , in some (other or the same) automaton we construct a transition t , labelled by α_i , with the input place $\tilde{q}$ and two output places: the place corresponding to the initial state q_i^0 of A_i and the place q' . If the state q' is terminal in its automaton, then the transition t has only one output place $\tilde{q}_i^0$.
4. For each two arcs $(q, q'), (p, p')$ labelled by mutually adjacent synchronization labels (these two arcs may be either in two different automata of the community, or in one automaton) we construct a transition t , labelled by τ with two input places $\tilde{q}, \tilde{p}$ and two output places $\tilde{q}', \tilde{p}'$. As before, if q' , or p' , or both are terminal states in their automata, then the corresponding output places for the transition t are omitted.

Figure 3 shows a Petri net, simulating the automata community Σ_1 , shown in Figure 1.

Fig. 3. A Petri net, simulating Σ_1 .

The observable behavior of a Petri net can be described in terms of Petri net languages [4].

Definition 5.2. Let PN be a labelled Petri net with transition labels from X . Let $\mathcal{R}_\emptyset$ be the set of all runs for PN , leading to the empty marking. A terminal language of PN is a set of all words (over X) labelling runs from $\mathcal{R}_\emptyset$.

Note, that in the definition of a terminal Petri net language an arbitrary fixed marking can be used as the target marking instead of the empty one. These two definitions are equivalent. The following theorem states, that for every automata community Σ there exists a Petri net $PN(\Sigma)$, simulating the behavior of Σ .

Theorem 5.3. For every automata community Σ there exists a Petri net $PN(\Sigma)$, such that the terminal language of $PN(\Sigma)$ coincides with the language $\mathcal{L}(\Sigma)$, accepted by the automata community Σ .

Proof sketch: It can be shown, that for each run of Σ there exists a corresponding run of $PN(\Sigma)$ and vice versa. The proof can be done by induction on the length of the run by the analysis of all cases of state transitions. $\square$

In the theory of Petri net languages it is known, that context-free languages are incomparable with Petri net languages [4]. More precisely, there exist context-free languages, which are not terminal Petri net languages. An example of such a language is the language $\{ww^R \mid w \in \{a,b\}^*\}$, where w^R denotes a word w in the inverse order, i.e. written from the right to the left. Then from theorem 5.3 we get the following

Proposition 5.4. There exist context-free languages, which are not accepted by any automata community.

The Petri net $PN(\Sigma)$ simulates runs of Σ not only as sequences of actions, but also as sequences of reachable states. These means, that the Reachability Problem for automata communities can be reduced to the Reachability Problem for Petri nets. Recall, that the Reachability Problem is to decide, given an initial state S and a finite state S' , whether there exists a run, leading from S to S' .

For Petri nets reachability is decidable [7], hence we obtain

Proposition 5.5. *Reachability Problem is decidable for interacting automata communities.*

References

1. R. Alur, S. Kannan, and M. Yannakakis. Communicating Hierarchical Automata. In *ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 169–178. Springer, 1999.
2. A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.
3. A. Finkel, and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256:1–2, pages 63–92, 2001.
4. M. Jantzen. Language Theory of Petri Nets. In *Petri nets: Central Models and their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 397–412. Springer, 1987.
5. R. Lanotte, A. M. Schettini, A. Peron, S. Tini. Dynamic Hierarchical Machines. In *Proc. of the Concurrency Specification and Programming (CSP'2000) Workshop. 7–9 October 2002. Vol.2*, Informatik-Bericht Nr.161, Humboldt-Universität zu Berlin, pages 205–216, 2002.
6. I. A. Lomazova. Nested Petri nets — a Formalism for Specification and Verification of Multi-Agent Distributed Systems. *Fundamenta Informaticae*, 43(1–4): 195–214, 2000.
7. E. W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM Journal on Computing*, 13:441–460, 1984.
8. W. Reisig. *Petri Nets. An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.
9. B.A. Trakhtenbrot. Automata, Circuits, and Hybrids: Facets of Continuous Time. volume 2076 of *Lecture Notes in Computer Science*, pages 4–23. Springer, 2001.
10. M. Yannakakis. Hierarchical State Machines. In *IFIP Theoretical Computer Science'2000*, volume 1875 of *Lecture Notes in Computer Science*, pages 315–330. Springer, 2000.