Cellular Resource-Driven Automata [⋆]

Vladimir A. Bashkin[1] and Irina A. Lomazova[2,3]

[1] Yaroslavl State University, Yaroslavl, 150000, Russia
bas@uniyar.ac.ru
[2] Higher School of Economics, Moscow, 101000, Russia
[3] Program Systems Institute of the Russian Academy of Science,
Pereslavl-Zalessky, 152020, Russia
i_lomazova@mail.ru

Abstract. Resource-driven automata (RDA) are finite automata, sitting in the nodes of a finite system net and asynchronously consuming/producing shared resources through input/output system ports (arcs of the system net). RDAs themselves may be resources for each other, thus allowing the highly flexible structure of the model. RDA-nets are expressively equivalent to Petri nets [2].
In this paper a new formalism of *cellular RDAs* is introduced – RDA-nets having an infinite (but regular) system net. A hierarchy of classes is built using restrictions on the underlying grid (in the 1-dimensional case), the expressive power of several major classes is studied.

1 Introduction

In [1] we introduced nets of active resources (AR-nets) – a formalism for modeling distributed systems from resource perspective, when active components and resources are not distinguished. AR-nets formalism is in some sense dual to ordinary Petri nets. In AR-nets the Petri net sets of transitions and places are united into a single set of *nodes*, but the set of arcs is partitioned into two separate sets of *input arcs* and *output arcs*. Each node may contain *tokens*. A token in a node may fire, consuming some tokens through input arcs and producing some other tokens through output arcs. So the same token may be considered as a passive resource (produced or consumed by an agent), or as an active agent (producing or consuming resources) at the same time.

In [2] we considered extending the syntax of AR-nets to Resource Driven Automata Nets (RDA-nets). A RDA-net is a two-level model [8]. A system level of the RDA-net is defined by a flat AR-net, an agent is defined by a finite state machine (automaton), being placed into a system node as a token. Agents interact through shared system resources; and an agent itself serves as a shared system resource. From the modeling perspective the system net in a RDA-net is a "map" of input and output ports (arcs), that connects different system nodes. The agent

[⋆] This research is partially supported by Russian Fund for Basic Research (projects 09-01-00277, 11-01-00737, 11-07-00549) and Higher School of Economics.

may use this ports to interact with system resources (produce/consume), residing in adjacent nodes. So an agent may natively produce/consume/copy/move other agents through these ports (or even use itself as a resource).

In this paper we present a natural generalization of RDA-nets, having an *infinite* regular grid. We call them cellular resource-driven automata (CRDA).

The paper is organized as follows. Section 2 contains some basic definitions and notions, including formal definitions of Resource-Driven Automata nets (RDA-nets). In Section 3 a motivating example of "cellular" behaviour is given, modeling Foraging Ants problem. In Section 4 cellular resource-driven automata are defined. Also a number of classes of CRDA are studied, having different expressive power. Section 5 contains some conclusions.

2 Basic definitions

Let S be a finite set. A *multiset* m over a set S is a mapping $m : S \to Nat$, where Nat is the set of natural numbers (including zero), i. e. a multiset may contain several copies of the same element.

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 and the union of two multisets m and m' are defined as usual: $\forall s \in S : m + m'(s) = m(s) + m'(s),\ m \cup m'(s) = max(m(s), m'(s))$.

By $\mathcal{M}(S)$ we denote the set of all finite multisets over S.

A Net of Active Resources (AR-net) [1] is a tuple $AR = (V, I, O)$, where V is a finite set of *resource nodes* (*vertices*); $I : V \times V \to Nat$ is a *consumption relation* (*input arcs*); $O : V \times V \to Nat$ is a *production relation* (*output arcs*).

In graphic form the nodes are represented by circles, the consumption relation by dotted arrows and the production relation by solid arrows (Fig. 1).

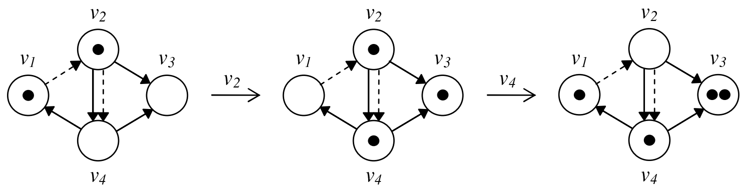


Fig. 1. Node firings in a net of active resources

A *marked net of active resources* is a pair (AR, M_0) where AR is an AR-net and $M_0 : V \to Nat$ is it's *initial marking*.

As usual, pictorially the marking is denoted by filled circles.

A resource node $v \in V$ is *active* in a marking M iff $M(v) > 0$ (the node v is not empty) and $\forall w \in V \ \ M(w) \geq I(w, v)$ (there are enough tokens in all it's input nodes). An active node v may *fire* yielding a new marking M' s.t.

$$\forall w \in V \ \ M'(w) =_{def} M(w) - I(w, v) + O(v, w).$$

The token may be producing, consuming, produced and consumed at the same time (through different incident arcs). It can be even self-copied or self-consumed. The syntax of AR-nets differs from the syntax of Petri nets. However, they define the same class of systems [1].

Resource Driven Automata Nets [2] are defined as AR-systems with tokens (resources) being a specialized extension of finite state machines, called resource driven automata. Thus we refine tokens in AR-nets by defining their own behavior. In the course of its running an automaton token may consume and produce resources-tokens from/to the nodes adjacent to the node it resides in. We label arcs in AR-net with port names to indicate nodes with accessible resources.

Let Ω be a finite set of *types*, $Const$ be a finite set of typed individual objects. We call these objects constants. The type of a constant $c \in Const$ is denoted by $Type(c)$. For $a \in \Omega$ by $Const(a)$ we denote the set of all constants of type a.

Let then Π be a finite set of typed (by elements of Ω) *ports* (port names). The type of a port $\pi \in \Pi$ is denoted by $Type(\pi)$.

An underlying AR-net with arcs labeled with ports is called a *system net.*

Definition 1. *[2] A* system net *is a tuple* $SN = (V, I, O, \pi)$*, where* (V, I, O) *is an AR-net,* $\pi : (I \cup O) \to \Pi$ *is a function, labeling arcs by port names.*

Definition 2. *[2] A* marking M *of the system net* SN *is a function*

$$M : V \to \mathcal{M}(Const),$$

that maps a multiset of objects to each node of the net.

A marked system net *is a pair* (SN, M_0) *where* $SN = (V, I, O, \pi)$ *is a system net and* M_0 *is it's* initial marking.

Denote by Var the set of variables typed with Ω-types. Let $a \in \Omega$. We define a language $L(a)$ of *resource transformation expressions* of type a as follows.

Let $\pi \in \Pi$. An *input term* is a term of the form $\pi?e$, where e is a variable or a constant of type $Type(\pi)$. Similarly, an *output term* is a term of the form $\pi!e$ with the same conditions on π and e. Now, a resource transformation expression is a term of the form $\alpha_1; \alpha_2; \ldots; \alpha_k$, where α_j $(j = 1, \ldots, k)$ is either an input or an output term (types of subexpressions $\alpha_1, \alpha_2, \ldots, \alpha_k$ may differ).

Definition 3. *[2] A* resource driven automaton *(RDA) of type* $a \in \Omega$ *is a tuple* $A = (S_A, T_A, l_A)$*, where* S_A *is a finite set of states,* $T_A \subseteq S_A \times S_A$ *is a transition relation, and* $l_A : T_A \to L(a)$ *is a transition labeling.*

Thus a RDA is just a finite state machine with labeled transitions and no distinguished initial state. Further in RDA-nets resource driven automata will play a role of tokens in AR-nets. In a special case, when a RDA contains exactly one state with no transitions, such a token is just a usual colored token without its own states and behavior. We call such tokens elementary resources.

Definition 4. *[2] Let* $\Omega = \{a_1, \ldots, a_k\}$ *be a finite set of types, and let* $\mathcal{A} = (A_1, \ldots,\ A_k)$ *be a finite set of RDAs, where*

1. *for a type $a_i \in \Omega$ the set $Const(a_i)$ is defined as a set of all states of RDA A_i; $Const =_{def} \bigcup_{a \in \Omega} Const(a)$;*
2. *each A_i is a RDA of type a_i over the set of types Ω, the set Var of variables typed with Ω-types, and the set of constants $Const$;*

An RDA-net $RN = (\Omega, SN, (A_1, \ldots, A_k))$ *consists of a finite set of RDA $(A_1, \ldots, A_k)$ with types from Ω as described above, and a system net SN over a set Ω of types, set $Const$ of constants, and some set Π of Ω-typed ports.*

A marking *(a state) in a RDA-net is, by definition, a marking in its underlying system net.*

By abuse of notation we will not differ automaton A and its name/type a. From now we will write $(a|s)$ for the constant denoting the state s of an automaton A of type a. Contextually we call a constant an *agent*, a *resource*, or just an *object*.

We define an interleaving semantics for RDA-nets. A run in a RDA-net is a sequence of agent transition firings. Thus only agents may change a state.

Now we are ready to define an agent's transition firing rule.

Informally speaking, firing a transition t in an agent a requires resources listed in input subterms of the resource transformation expression, labeling t. Input subterm $p?e$ describes a resource e, which should be obtained via a port p in the system net. A source node for this port arrow in a system net is a node, where this resource should be taken from. Similarly, if possible a firing of an automaton transition consumes required resources and produces new resources in line with output subexpressions of the transition label.

As usual, a mode of transition firing is determined by a variable binding.

Definition 5. *[2] A* binding *of the variables is a function* **bind** : $Var \to Const$ *such that for all $\varphi \in Var$ we have $Type(\mathbf{bind}(\varphi)) = Type(\varphi)$.*

Let M be a marking in a RDA-net $RN = (\Omega, SN, (A_1, \ldots, A_k))$, $a|s$ – a RDA in a state s residing in a node v of RN in M. Let b be a variable binding. A transition $t = (s, s')$ with a label $l_a(t)$ is *enabled* in M iff there is a one-to-one correspondence between input subterms in $l_a(t)$ and objects in M s.t. for a subterm $p?e$ there is an object $e[b]$ in a node v' of RN in M with an input arrow (v', v) labeled by p in RN.

The correspondence described above defines a 'submarking' $\check{M}$ mapping a node v of RN to the multiset of objects residing in v and singled out by input subterms of $l_a(t)$. It includes all objects, consumed by firing of t with binding b. Note, that for given t and b marking $\check{M}$ is defined nondeterministically. By $\mathbf{in}(t[b])$ we will denote the set of all such markings.

Similarly we define a set $\mathbf{out}(t[b])$ of markings mapping a node v to objects produced by transition t with a binding b in line with output subterms of $l_a(t)$.

Definition 6. *[2] Let $(a|s)$ be an agent, residing in a node $v \in V$ by a system marking M, and $t \in T_a$ be a transition with $t = (s, s')$.*

A transition t is enabled *with a variable binding b iff there exists a marking $\check{M} \in \mathbf{in}(t[b])$ s.t. for each $u \in V : \check{M}(u) \subseteq M(u)$.*

An enabled binded transition may (nondeterministically) fire to a new marking M' s.t. for each node $u \in V : M'(u) = M(u) - \check{M}(u) + \hat{M}$, where $\check{M} \in \mathbf{in}(t[b])$ and $\hat{M} \in \mathbf{out}(t[b])$.

RDA nets allow to model different aspects of systems with resources (Fig. 2).

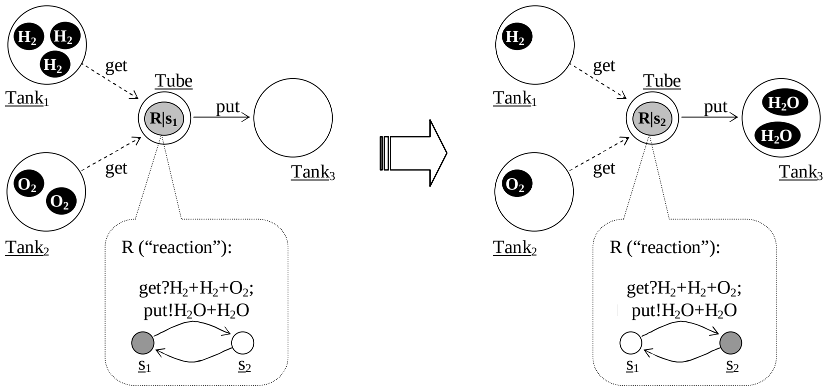


Fig. 2. RDA-net example: chemical reaction

Here we can see a process of water syntheses. The only active agent (RDA) of this system – the automaton R – defines the chemical reaction proper ("two molecules of hydrogen and one molecule of oxygen transmute into two molecules of water"). The system net represents a physical environment: three reservoirs and a tube, where the reaction takes place. Ports *get* and *put* link these two levels (or two aspects) of the system. From the system's point of view these ports are physical gates, from the agent's point of view they are parameter names.

The system level of the RDA-net is a labeled net of active resources, i.e. a labeled oriented graph with two types of arcs – input and output ports. However, the semantics is different. In AR-nets incident arcs define the resources that *must* be consumed and produced by an agent, situated in the node. In RDA-nets they identify places, that *may* be used as input or output storages by the agent.

By definition RDA-nets represent a formalism of local finite transformations of multisets without zero-testing. So it is quite easy to see that

Theorem 1. *[2] RDA-nets are expressively equivalent to Petri nets.*

3 Motivating example: Foraging Ants

The next model describes a well-known problem in artificial life (see e.g. [6]). Ants wander randomly looking for sources of food. They are able to determine directions: North, East, South and West. If they find food, they leave behind a pheromone trail. Or, if they find a pheromone trail, they follow it in search of food. This is a classical example of decentralized coordination. We will use:

Types:

item	(* something on the ground *)
ant	(* active agent (automaton) *)

Constants:

fn, fe, fs, fw : item	(* pheromones: food to the north/east/south/west *)
nfn, nfe, nfs, nfw : item	(* absence of corresponding pheromones *)
food : item	(* some eatable thing *)
a_1, a_2, ..., a_s : ant	(* a certain finite number of ants *)

Ports:

?get, !put : item	(* "can be taken/put" *)
?Leave : ant	(* "can go away" *)
!North, !East, !South, !West : ant	(* "can go North/East/South/West" *)

For the sake of shortness we will also use four macros:

MoveW ::= Leave?this; West!this	MoveN ::= Leave?this; North!this
MoveS ::= Leave?this; South!this	MoveE ::= Leave?this; East!this

Here *this* is a macro itself. It denotes the current automaton in the current state. For example, the macro $\big[Leave?this; North!this\big]$ on the transition from the state $\underline{ate}$ to the state $\underline{markingN}$ of the automaton a_i defines the expression $\big[Leave?(a_i|\underline{ate}); North!(a_i|\underline{markingN})\big]$. Informally, with this transition the automaton "transfers" itself from one place to another (by some system ports).

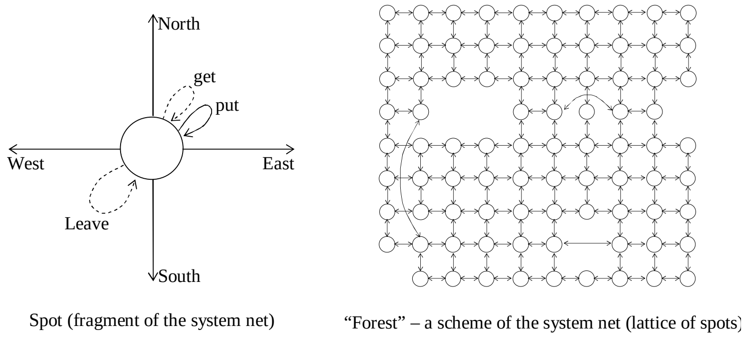

North
get
put
West
East
Leave
South
Spot (fragment of the system net)
"Forest" – a scheme of the system net (lattice of spots)

Fig. 3. Foraging ants — system net example

A system net represents a topography scheme of the "forest" (Fig. 3). It consists of a finite number of spots (locations), organized into a kind of a lattice (possibly incomplete). Each spot can be connected with at most four of its "neighbors" by four kinds of output ports: *North*, *East*, *South* and *West*. The ants can use these ports to leave the spot. To model the exit of the ant from the current spot we also use a special input port *Leave*. There are also two ports *get* and *put* that allow taking and putting an item, laying on the ground, by the ant.

Initially, each spot contains a set *nfn*, *nfe*, *nfs*, *nfw* of markers ("there are no pheromones"), a finite (possibly, empty) set of ants and a finite (possibly, empty) set of food items.

An ant is modeled by an automaton with 14 states. Its kernel consists of only two states: idle and ate. At the state idle it has three general behaviors: eat the food (if it exists in the spot); move randomly in one of four directions; follow the existing pheromone trail (also in one of four directions). At the state ate the ant prepares to start a new pheromone trail, moving randomly into a neighboring spot. If it moves to the northern spot, then the ant will continue moving to the north, marking its trail by "sf-sf-sf-…"; if it moves to the east, then "wf-wf-wf-…", etc.

Consider a scheme, containing only kernel states idle and ate and their incident outgoing transitions (Fig. 4).

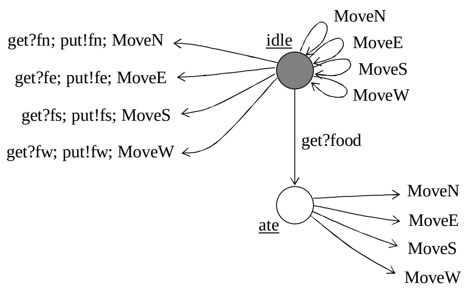


Fig. 4. Foraging ants — object automaton (kernel fragment)

Now consider the remaining part of the ant automaton. It consists of four similar fragments, corresponding to the direction of the first move. Here is a northern one (Fig. 5).

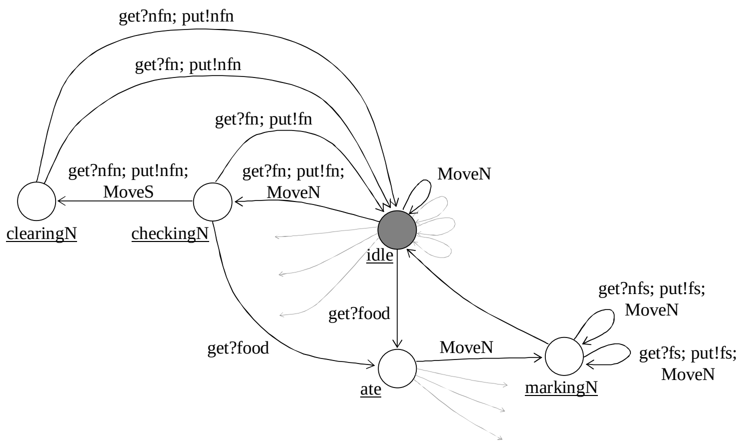


Fig. 5. Foraging ants — object automaton ("Move to the North" fragment)

It contains three additional states: checkingN, clearingN and markingN.

The ant is in checkingN state, if its previous move was to the north, following the pheromone *fn*. If the food really exists in this spot, it can be eaten (transition to the state ate). If the pheromone trail is continued in this spot, the ant can calmly return into the state idle. But if there is no pheromone *fn* in this spot (*nfn* item), then the ant can move back to the southern spot and erase the pheromone *fn* (state clearingN).

The ant is in markingN state, if its previous move was to the north in the state ate. Then the ant makes a random number of moves to the north, marking the spots with a pheromone *fs*.

4 Cellular RDAs

Real world is an infinite (or rather big) "forest". Consider finite communities of resource-driven automata, residing in the nodes of some infinite (but regular) system net. These systems resemble cellular automata (CA) with their infinite grid and finite set of rules for the individual cell, so we will call the new formalism *cellular RDA* (CRDA). More precisely, CRDA are close to asynchronous CA, where individual cells are updated independently. It is known that asynchronous CA are expressively equivalent to synchronous CA [10].

However, there is a substantial difference between CRDA and CA: the individual cell contains a multiset of tokens (active RDAs or simple resources), hence even the one-cell CRDA can have possibly infinite number of states. Moreover, we have a more flexible syntax of node interconnections (input and output ports).

Consider two different kind of initial marking:

1. *infinite CRDA:* all cells of the infinite grid except the finite subset are marked by the same set of resources;
2. *finite CRDA:* all cells of the infinite grid except the finite subset are empty.

It is known that cellular automata are Turing powerful. In CRDA the expressive power of a single cell is higher than in classical CA: it is not a finite automaton but a Petri net. Hence we easily obtain the next theorem:

Theorem 2. *Infinite CRDA are Turing powerful.*

Proof. Consider some asynchronous cellular automaton. The firing rule of an individual cell is actually a finite-state machine M, reading data from the neigbouring cells. So we just encode this rule into an RDA A^M and put a copy of A^M to every cell of the grid.

Finite CRDA provide a different (in some sence more Petri-like) modeling approach: the set of computing devices (marked cells) is always finite. In general this set may grow in two ways: in width (the number of cells) and in depth (the number of tokens). Obviously, the first type of growth ("widening") is necessary for universal computations, because otherwise we obtain a finite grid and hence an RDA-net (a Petri net). But is it enough?

In this paper we study the expressive power of some simple classes of finite CRDA. Note that Cellular Automata are Turing powerful even in one-dimensional case (M.Cook, S.Wolfram), where the grid is a row of cells. Hence we also consider only 1-dim CRDA.

The hierarchy of classes will be defined by the topology of the cell neighbourhood – the set of available adjacent ports. The most general case is depicted in Fig. 6 (the corresponding class is called *1-dim CRDA*). Here active tokens (RDAs), situated in the cell, can consume/produce resources from/to three cells – left neighbour ("West"), right neighbour ("East") and the current cell. There are no restrictions on tokens.

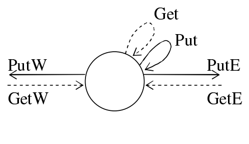
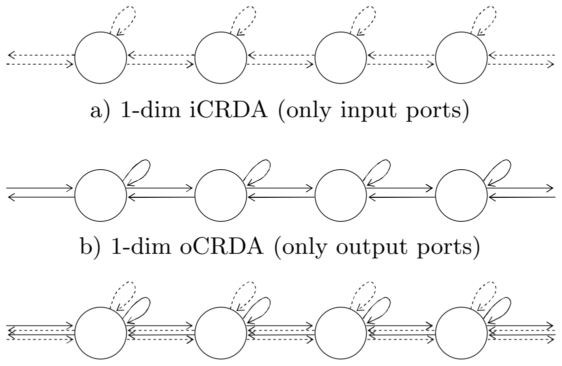


Fig. 6. The complete set of individual ports

In this paper we consider three most fundamental classes of the hierarchy, induced by the topology of the 1-dim grid. They are: (1) 1-dim CRDA with only input ports (1-dim iCRDA), (2) 1-dim CRDA with only output ports (1-dim oCRDA), (3) 1-dim CRDA. The corresponding grids are depicted in Fig. 7.



a) 1-dim iCRDA (only input ports)

b) 1-dim oCRDA (only output ports)

c) 1-dim CRDA (all ports)

Fig. 7. Considered grids

Theorem 3. *1-dim iCRDA = finite automata.*

Proof. It is easy to see that in 1-dim iCRDA the set of non-empty cells cannot grow (the active token cannot put tokens to the neighbouring cells, because there are no output ports). Moreover, it cannot put tokens into it's own cell – hence this cell is bounded.

Recall that Communication-free nets (also called BPP-nets) are Petri nets in which every transition has exactly one input place (cf. [5]). They are closely related to Basic Parallel Processes [3], which form a subclass of Milner's CSS.

Lemma 1. *For each oCRDA there exists bisimilar communication-free net.*

The proof can be found in the Appendix.

Theorem 4. *1) Communication-free Petri nets $\subseteq$ 1-dim oCRDA;*
2) Petri nets $\not\subseteq$ 1-dim oCRDA;
3) 1-dim oCRDA $\subset$ Turing machines.

Proof. (1) The modeling of CF-net by 1-cell oCRDA is obtained as follows. We introduce a single RDA with states made from places of the net. For each transition of the net we introduce a transition in this RDA from the input place (corresponding control state) to some of the output places (control state). All additional output arcs are substituted by output expressions ($put!\cdot$), ascribed to this transition, producing the corresponding numbers of RDAs in the corresponding control states.

In the initial marking we just replace each placed token by the RDA in the corresponding control state, positioned into some cell of the grid.

(2) Follows from: the decidability of bisimulation for communication-free nets [4]; the undecidability of bisimulation for general Petri nets [7]; Lemma 1.

(3) Follows from (2) and non-universality of Petri nets.

Theorems 3 and 4 show that:

1. In CRDA output ports are in some sense more expressive than input ones.
2. Only input ports are too weak to effectively use the infiniteness of the grid: we can restrict the workspace to a finite number of cells. In fact, to a single cell – it is easy to see that 1-cell iRDA are also equivalent to finite automata.
3. Only output ports also cannot use the infiniteness of the grid, but only modulo bisimilarity: we can restrict the workspace to a single cell without harming the observable behaviour.

On the other hand, the total set of links (ports) is universal:

Theorem 5. *1-dim CRDA are Turing powerful.*

The proof can be found in the Appendix.

5 Conclusion, Future Work

In this paper we have presented a new formalism of cellular RDA-nets for multi-agent and cellular systems modeling. The syntax of this formalism allows to combine the resource-processing of Petri Nets with the spatial dynamics of Cellular Automata.

In the paper we proved that 1-dim oCRDA contain communication-free Petri nets and do not contain general Petri nets. An interesting open problem is whether 1-dim oCRDA are equal to CF PN.

We considered only three basic classes of hierarchy, induced by the restricted topology of 1-dimensional grid. There are a lot of other interesting classes that can be studied (abbreviations define possible ports): $\{GetW, Get, Put, GetE\}$, $\{PutW, Get, Put, PutE\}$, $\{Get, Put, GetE, PutE\}$, $\{Get, Put, PutE\}$, and so on. It seems that different sets of ports may produce specific coordination patterns with specfic expressive properties.

The other possible way of classification is the restriction of automata structure/behaviour. For example, we can structurally bound the maximal number of tokens residing in the cell. In other words, doing this we allow only one way of growth – "in width", but not "in depth". This may be even more meaningful in the context of *cellular* models, than our original finite CRDA syntax definition (but less Petri-net-like). Moreover, such a restriction will not reduce the expressive power of iCRDA and CRDA, since the nets, presented in the proofs, are already cell-bounded. For oCRDA it is not quite clear.

The other possible direction of research are infinite CRDA (where all cells of the grid are initially filled by some "sleeping" automata). This kind of syntax seems to be more useful as a model of physical reality (simulation of chemical, biological, physical and sociological processes). Also note that, in contrast to Cellular Automata, CRDA allow to mark individual cell by possibly *infinite* number of colours. Hence the modeling principles may be quite different.

The other possible direction of research are n-dim CRDA with different grid topologies and CRDA with non-regular grids.

References

1. V.A. Bashkin. Nets of active resources for distributed systems modeling. Joint Bulletin of NCC&IIS, Comp. Science. Novosibirsk. 2008. V.28. P.43–54.
2. V.A. Bashkin, I.A. Lomazova. Resource Driven Automata Nets. Fundamenta Informaticae, 2011. V.109(3). P.223–236.
3. S. Christensen. Decidability and Decomposition in Process Algebras. PhD thesis, Edinburgh University, 1993.
4. S. Christensen, Y. Hirshfeld, F. Moller. Bisimulation equivalence is decidable for Basic Parallel Processes. Proc. of CONCUR'93, LNCS 715, 1993.
5. J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes, Fundamenta Informaticae, 1997. V.31. P.13–26.
6. D. Goldin, D. Keil. Indirect Interaction in Environments for Multi-agent Systems. Environments for Multi-Agent Systems II, LNCS 3830, 2005. P.68–87.
7. P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. *Proc. of STACS'94*. LNCS 775, 581–592, 1994.
8. I.A. Lomazova. Nested Petri Nets – a Formalism for Specification and Verification of Multi-Agent Distributed Systems. Fundamenta Informaticae. 2000. V.43:195–214.
9. M. Minsky. Computation: Finite and Infinite Machines. Prentice Hall, 1967.
10. C.L. Nehaniv. Asynchronous Automata Networks Can Emulate Any Synchronous Automata Network. Int.J. of Algebra and Computation. 2004. V.14(5-6). P.719–739.

6 Appendix

Proof of Lemma 1. First note that in the situation with no input ports an agent (active token) cannot observe the contents of the three adjacent cells. The only information the token has is its own control state, so its behaviour does not depend on its location (cell). Hence we can replace all *PutW*- and *PutE*-expressions by *put*-expressions and this will not change the overall observable behaviour of the system (by observable behavior of the system we mean the observable behaviour of its active component — resource driven automata).

So 1-cell oCRDA are bisimulation equivalent to 1-dim oCRDA, and hence it is sufficient to prove that for any 1-cell oCRDA there exists bisimilar CF-net.

The simulation of 1-cell oCRDA by CF-net is obtained as follows:

1. For each state s of each RDA a we introduce a place $(a|s)$ in the net.
2. For each transition $t = (s, s')$ of each RDA a we introduce a transition t in the net with an input arc $\big((a|s), t\big)$ and an output arc $\big(t, (a|s')\big)$. Additionally, for each output expression $put!(b|u)$ on the transition t in RDA a we introduce an output arc $\big(t, (b|u)\big)$.
3. The initial marking is obtained as follows: we put into each place $(a|s)$ exactly the initial number of RDAs a in the control state s.

It is easy to see that a natural one-to-one mapping between markings of oCRDA and CF-net is a bisimulation: whenever a transition is possible on the one side, the corresponding transition with the same label is possible on the other side.

Proof of Theorem 5. We will show that for any given Minsky machine [9] (two-counter automata with zero-testing) it is possible to construct an equivalent CRDA.

A Minsky machine with counters c_1 and c_2 is a sequence of commands

$$1: \; COMM_1; \quad 2: \; COMM_2; \quad \dots \; ; \quad n: \; COMM_n$$

where $COMM_n$ is a $HALT$-command and $COMM_i$ $(i = 1, 2, \dots, n-1)$ are commands of the following two types (assuming $1 \le k, m \le n, 1 \le j \le 2$)

1) $i: \; c_j := c_j + 1; \; goto \; k$
2) $i: \; if \; c_j = 0 \; then \; goto \; k \; else \; (c_j := c_j - 1; \; goto \; m)$

The simulating 1-dim CRDA is constructed as follows.

The grid (the row of cells) is composed from three parts: infinite left half-row; central cell; infinite right half-row. The central cell is marked by a distinguished token `C`. The value of the first counter (c_1) is stored in the left ("western") half-row: there are exactly c_1 cells to the left of the central one, marked by token • (each cell is marked by a single copy of •; there are no empty cells between marked). To the left of the last •-marked cell there is a single "left border" cell, marked by special token `W`.

Similarly, the value of the second counter (c_2) is stored in the right ("eastern") half-row, the only difference is in the marking of the border – it is not `W`, but `E`.

The described structure will be maintained during all computations of the CRDA. For example, increment of c_1 means increasing the length of the left half-row by 1, decrement – decreasing the length by 1, and so on.

The program of the Minsky machine is modeled by a single resource-driven automaton `Auto`, situated in the central cell. It has n states, corresponding to all commands of the program (initially it is in its first state), and a number of transitions: one transition for each increment command and two transitions for each conditional decrement command.

Consider commands for the first counter in details.

1) $i:\ c_1 := c_1 + 1;\ \mathit{goto}\ k$

This command is modeled by a transitions from state i to state k:

$$i \to \big[\mathit{Get?}(\texttt{Auto}|i);\ \mathit{PutW!}(\texttt{IncW}|k)\big] \to k.$$

Note that `Auto` is erased, and a new automaton `IncW` is put into the neigbouring left cell. The new automaton saves the value k as its internal state.

Automaton `IncW` has n states, each state has two loops. Considering state k, the loops (transitions from k to k) are:

$$k \to \big[\mathit{Get?}\bullet;\ \mathit{Get?}(\texttt{IncW}|k);\ \mathit{PutW!}(\texttt{IncW}|k);\ \mathit{Put!}\bullet\big] \to k;$$
$$k \to \big[\mathit{Get?}\texttt{W};\ \mathit{Get?}(\texttt{IncW}|k);\ \mathit{PutW!}\texttt{W};\ \mathit{Put!}\bullet;\ \mathit{PutE!}(\texttt{MoveE}|k)\big] \to k.$$

Automaton `MoveE` has n states, each state has three loops. Considering state k, the loops (transitions from k to k) are:

$$k \to \big[\mathit{Get?}\bullet;\ \mathit{Get?}(\texttt{MoveE}|k);\ \mathit{Put!}\bullet;\ \mathit{PutE!}(\texttt{MoveE}|k)\big] \to k;$$
$$k \to \big[\mathit{Get?}\texttt{W};\ \mathit{Get?}(\texttt{MoveE}|k);\ \mathit{Put!}\texttt{W};\ \mathit{PutE!}(\texttt{MoveE}|k)\big] \to k;$$
$$k \to \big[\mathit{Get?}\texttt{C};\ \mathit{Get?}(\texttt{MoveE}|k);\ \mathit{Put!}\texttt{C};\ \mathit{Put!}(\texttt{Auto}|k)\big] \to k.$$

The first loop "moves" the automaton `MoveE` from the left border of the row to the center. The third loop may fire only in the central cell: it erases `MoveE` and restores `Auto`. The purpose of the second loop will be seen later.

2) $i:\ \mathit{if}\ c_1 = 0\ \mathit{then\ goto}\ k\ \mathit{else}\ (c_1 := c_1 - 1;\ \mathit{goto}\ m)$

This command is modeled by two transitions: from state i to state k and from state i to state m:

$$i \to \big[\mathit{GetW?}\bullet;\ \mathit{Get?}(\texttt{Auto}|i);\ \mathit{PutW!}\bullet;\ \mathit{PutW!}(\texttt{DecW}|k)\big] \to k;$$
$$i \to \big[\mathit{GetW?}\texttt{W};\ \mathit{PutW!}\texttt{W}\big] \to m.$$

Note that the second transition may occur iff the first counter is empty (the token `W` is "visible" by the automaton `Auto`). Here we model zero-testing by "looking to the left (cell)".

Automaton `DecW` has n states, each state has two loops. Considering state k, the loops (transitions from k to k) are:

$$k \to \big[\mathit{Get?}\bullet;\ \mathit{Get?}(\texttt{DecW}|k);\ \mathit{PutW!}(\texttt{DecW}|k);\ \mathit{Put!}\bullet\big] \to k;$$
$$k \to \big[\mathit{Get?}\texttt{W};\ \mathit{Get?}(\texttt{DecW}|k);\ \mathit{GetE?}\bullet;\ \mathit{PutE!}\texttt{W};\ \mathit{PutE!}(\texttt{MoveE}|k)\big] \to k.$$

The automaton `MoveE` is already defined.

The second counter is modeled in a similar way, we just replace letter `W` by letter `E` and vice versa.