This article was published in "Proceedings of International Conference Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration and Engineering. KIMAS-03". Boston: IEEE Boston Section. 2003, p. 65-70.

Automata Theory for Multi-Agent Systems Implementation

Lev Naumov, Anatoly Shalyto
Saint-Peterburg State Institute of Fine Mechanics and Optics, Computer Science Department, 197101 Sablinskaya st. 14, Saint-Peterburg, Russia
levnaumov@mail.ru
http://is.ifmo.ru

Abstract— *Work shows that automata theory can be useful for engineering, developing and documenting multi-agent systems. Algorithmization and programming principles for logic control are formulated, regarding algorithms and programs as finite automata.*
This work was performed with the help of Russian Fund of Fundamental Investigations according to grant 02-07-90114..

1. INTRODUCTION

Finite automata, which in the past were mainly used in hardware, are presently finding extensive application in programming. We outline the basic algorithmization and programming principles for logical control [1-2], reactive systems [3-5], object-oriented tasks [6] and calculating algorithms [7] regarding programs as finite automata. Besides the traditional fields of application, such as compiler design, finite automata are presently used in programming programmable logic (PL) controllers, describing the behavior of certain objects like in object-oriented programming, as well as in programming protocols, games, and PL circuitry [7].

We would like to dispel the misbelief that "automata are becoming obsolete" and show they are just beginning to find extensive reliable application in programming. So let us offer a new method of systems engineering, developing and documenting based on usage an automaton as a primary abstraction. We called this approach the switch-technology (state-technology or, more exactly, automaton-technology) [1-8].

We can say that each automaton, developed in the context of this technology possess the properties and characteristics of agents [9,10]. It can be declared though the concept of "agent" cannot be defined rigorously. But nevertheless they can be associated with each other because automata in switch-technology are also the hardware or software goal-oriented substances [10]. Our automata are also autonomous, can communicate with others, can keep an eye on environment (realize reactive systems), able to use knowledge, can be adoptive and can study [10]. They can incarnate the same tasks as combinational, consecutive, task-oriented and task-selective agents [9]. But, in contrast to agents can be provided with formal definition and a lot of formal methods for developers.

Switch-technology automata can use any other programming approaches inside their envelope. For example, for decision making they can use neural networks, genetic algorithms or other.

Complex multi-agent systems in the context of suggested technology are to be realized as a systems of communicating automata.

2. SWITCH-TECHNOLOGY PARADIGM FOR LOGICAL CONTROL TASKS

The International Standard IEC 1131-3 establishes programming languages for PL controllers and PC-controllers-industrial (control) computers (usually IBM PC compatibles) with SoftPLC and SoftLogic application software. There also exist programming languages for microcontrollers and industrial (control) computers. According to [1], thus far no algorithmization language for logic control problems (based on true and false logic) has been developed that might be helpful in

- understanding what has been done, what is being done, and what must done in a programming project in different fields of knowledge,
- formally and isomorphically converting an algorithm into programs in different programming languages with a minimal number of internal (control) variables, because these variables hinder clear understanding of programs,
- easily and correctly modifying algorithms and programs, and
- correctly certifying programs
- producing reliable products.

Since such a language is not available, thus far there is no

algorithmization and programming technology that might enhance the quality of software for logic control systems.

Review [1] surveys the well-known algorithmization and programming technologies for logic control and reactive systems, i.e., the technologies underlying the new switch-technology meeting the above requirements, which can also be called the state-technology or, more exactly, the automaton-technology. We describe its basic principles.

1. “Internal state” (simply, state in what follows) is the basic concept in this technology.

State is regarded as a kind of abstraction introduced in algorithmization via one-one correspondence of a state with one of the physical states of the controlled object, because “operation of a production system is usually manifested as variations in its states”. Each state in the algorithm maintains the object in the respective state and a transition to a new state in the algorithm corresponds to a transition of the object to the respective state, thereby implementing the logic control.

The relationship of the states with internal (control) variables is manifested at the state coding stage, which is absent in traditional programming. The number of control variables depends on the coding scheme.

The approach used in automaton theory is essentially different from the approach usually used in programming, in which internal (usually, binary) variables are introduced, if necessary, in the course of programming and then each set of code values is declared a state of the program. Since the “state” concept is usually not used in application programming, the number of states in a program containing, for example, n binary interval variables in most cases is obscure. However, the number of states may vary from n to 2^n . It is also not clear where do these variables come from, how many variables are there, and the purpose for which they are applied. In cyclic implementation (due to output-input feedback), a program may also operate sequentially even in the absence of control variables [1].

2. Along with the state concept, the concept of “input” naturally requires the concept of an “outputless automaton” (outputless automaton = states + inputs). If the concept of an “output” is defined, by which we mean “action” and “activity”, this formula reads automaton = states + inputs + outputs. The corresponding programming field can be called the “automaton programming”, and program designing can be called the “automaton program designing”. Action is single-time, instantaneous, and continuous, whereas an activity may last long and interrupted by some input. An input in general may change the state and initiate output with or without change of state.

3. The main model underlying the automaton technology is a “finite deterministic automaton” (in the sequel, simply automaton), which is an automaton with internal states, but not an automaton with behavior functions [1].

4. Automata without output converters (Moore machines), Mealy machines, and combined machines (C-machines or Moore-Mealy machines) are used as structural models. The main structural model is defined by Moore machines, in which state codes and output values are distinct and the values of output variables in each state do not depend on inputs. This simplifies the introduction of changes in their description. In [1], “controllability” is defined to be the properties of algorithms and programs that aid in their correction. Initially, the number of states of a Moore machine can be chosen equal to the number of states of the controlled object (including its faulty states, if necessary). Subsequently, additional states related, for example, to operator's faults may be included. Then the number of automaton states can be minimized by combining equivalent states or using some other structural model and this, unless absolutely unavoidable, is rather undesirable.

In a Moore machine, the values of output variables are retained in the memory realizing these states as long as the machine exists in the corresponding state. In a Mealy machine, these values are also formed in the corresponding transition and, consequently, an additional memory different from the state-realizing memory is required for long storage of these values. In this respect, Moore and Mealy machines are not equivalent.

5. Automata of the first and second kind are also used [1]. Preference is given to the latter, in which a new state and output values are formed without delay in a “cycle” (in one program cycle).

6. The automaton states are encrypted by different coding schemes, for example, forced, free, binary, binary logarithmic, and multivalued codes. Multivalued state coding is preferable, because it can be used for the states of automata (due to the presence of only one active state in them) and the traditional viewpoint that automata are particular Petri networks can be discarded, because a single variable cannot be used for state coding for the reason that several states are concurrently active in a Petri network. While only one multivalued variable is suficient to distinguish the states of an automaton regardless of the number of states, the number of binary variables required to distinguish the positions equals number of states in a Petri network in which not more than one label can exist in each position (analog of the Standard NFC-03-190 Grafset). As shown in [1], a Grafset diagram with parallel “segments” can be replaced by a system of interconnected transition graphs. In the first case, while the number of binary variables is equal to the number of positions, the number of

multivalued variables for the second case is equal to the number of transition graphs, irrespective of the number of vertices in them.

7. The nonprocedural visual formalism of automaton theory, such as transition graphs (state charts or state transition charts) is used in programming the algorithmic model of finite-memory automata. In the names of these terms, as in automaton technology, preference is given to the concept of "state", rather than to the concept of "event" commonly used in modern programming. In our approach, the concept of an event is only secondary and, along with the input variable, is regarded as a modification of the input that may change the state.

The procedural visual formalism (graph-schemes of algorithms and programs) elaborated in theoretical programming does not use the "state" concept in explicit form and this complicates the understandability of this concept. This concept is also not used in the language of regular expressions of event algebra.

It is easy to recognize transition graphs as they are planner and have no height (like algorithm schemes, SDL- , and Grafset diagrams). They are far more compact compared to equivalent schemes consisting of functional blocks (traditional logic elements) and easily comprehensible, because interaction between transition graphs is implemented with data and the interaction between schemes is implemented by control.

8. An advantage of transition graphs, which must be "maximally" planar, is that every arc shows not all inputs (as minterms), but only those that ensure transitions along this arc. The inputs on every arc can be combined into Boolean formulas of arbitrary depth. Therefore, the description of the algorithm is highly compact.

The use of Boolean formulas in a model, as in hardware realization (structural synthesis of series circuits), widens the classical abstract finite automaton model, in which arcs are labeled only with the input alphabet characters, and aids in "parallel" processing of inputs. Compared to input-sequential program realization, such a description of algorithms in general decreases the number of states in automata generating these algorithms.

We can assert that every state in a transition graph "identifies" from the set of all inputs only the subset that ensures transitions from this state, i.e., decomposes the input set. Therefore, transition graphs can be applied in solving logic control problems containing a large number of inputs, thereby simplifying comprehensive testing along all routes.

9. Every transition graph must be semantically and syntactically correct. The first property determines whether the correct model has been designed, whereas the second property determines whether the model has been designed properly. In verifying the syntactical correctness, a transition graph is tested for attainability, completeness, consistency, realizability, and absence of generating circuits. In testing attainability, the presence of at least one path from every vertex to any other vertex is verified. Completeness is verified for every vertex and is useful in discarding loop labels, in particular, for Moore automata. If every vertex is consistent, priorities can be used instead of orthogonalization to reduce the program realization complexity. Realizability is ensured by different identically labeled vertices. Generating circuits [1] are eliminated via orthogonalization of labels of arcs forming the corresponding circuits.

10. Further refinement of the abstract finite-automaton model and output "parallelism", as in hardware implementation, are attained by indicating on vertices and/or arcs the values of output variables (activities) formed at these graph components. Display of the values of all output variables at each component (depending on the automaton structural model) aids in understanding the algorithm described by a transition graph due to the increased number of vertices. The transition graph thus constructed defines a structural finite automaton without any omissions of the values of output variables. For the transition graph of a Moore automaton, the values of output variables that do not change in successive transitions and shown at vertices at which transitions occur must be commented out to reduce the program size.

The finite-automaton model can be improved further by (at the cost of understandability) by indicating both the values of output variables and Boolean formulas (automaton formulas for embedded automata) at graph components [1].

11. Input and output "parallelism" aids in realizing parallel processes even by a single automaton, which, by definition, is a sequential state machine (only one vertex of the corresponding transition graph is active at an instant).

12. Unlike a memoryless automaton, the behavior of a finite-memory automaton depends on its prehistory: every transition to a state depends on the preceding state, whereas outputs of a Moore machine depend only on the state in which the machine exists.

In the automaton technology, these properties of finite-memory automata must be preserved and the transition graphs must not contain any ags and omissions to eliminate the dependence on state and output prehistory, respectively.

13. In formal and isomorphic transformation of a transition graph without ags and omissions to a program, the graph is

the specification test for the program. Here there is a possibility for certification, beginning from the convolute of every transition graph with its isomorphic program fragment. This approach is used to verify the attainability of implicitly defined states of a system.

14. One internal variable of significance digits equal to the number of states is sufficient for coding the states of a finite-state automata with multivalued codes.

Since (in principle) any logic control algorithm can be realized by one transition graph, the graph can be realized by a program containing only one internal (control) variable, regardless of the number of vertices in the graph.

In every transition, the previous value of the multivalued variable is automatically discarded and this variable is unrivaled.

Such a coding scheme can be applied only if the number of states of the automaton are known at the beginning of the algorithmization process.

15. Almost all programming languages, even the exotic functional block language, can handle multivalued variables.

A transition graph with multivalued coding for vertices corresponding to the automaton model is formally and isomorphically realized by one or two switch constructs of C [1] or their analogs in other programming languages. This explains the name of the technology. Moreover, the word "switch" is associated with the switching circuit theory-the base of logic control theory.

Transition graphs can be realized via two approaches, which can be called the "state-event" and "event-state" approaches. In the first approach, a transition graph is realized by one or two "state" switch constructs, whereas in the second approach, each "event" is associated with a function described by a "state" switch defining the transitions initiated by this "event". If events are represented by the values of a multivalued variable (function forming these values), then the state switch embedding event switches is the primary construct in the first approach, whereas the event switch embedding state switches is the primary construct in the second approach [1]. In the first approach, programs are designed via logic control principles (state-state transition under the action of an event) to make programs user-friendly. In advanced disciplines (for example, physics), the concept of "state space" is fundamental, whereas the concept of "event flow" is secondary. For example, (liquid, solid, gas) states of water are decisive, whereas state-state transition conditions and water interactions are secondary.

16. The value of a multivalued variable is characterized by the "position" of the program realizing the transition graph in the state space. Therefore, the concept of "observability" (for one internal variable) can be introduced in programming, regarding the program as a "white box" with one internal parameter.

17. From the foregoing, it follows that our approach is helpful in solving the problem, called decoding or recognition or identification of automata, because an automaton is recognized (decoded, identified) as soon as its transition graph is constructed. An automaton is not recognizable if it is an "absolutely black box" for which no information on its internal states is available. For finite-memory automata, "input-output" tests used in designing logic control systems are helpless in recognition and guaranteeing a predefined behavior of a system.

A finite deterministic automaton can be recognized by its "input-output" pattern if the maximal number of states is known in minimal (number of states) form and its transition graph is strongly connected. An automaton with known estimate of the number of states is called a "relatively black box".

18. Thus, it is not possible to recognize automata via testing algorithms and programs by their input-output patterns if the "state" concept is not defined. There is no problem of recognition in the automaton technology, because the state concept is defined and transition graphs are strongly connected. In the switch-technology, "black boxes" are discarded, but "white boxes" are used. An automaton defined in any form different from a transition graph without ags and omissions is called the "relatively black box" and an automaton defined by a transition graph without ags and omissions is called the "absolutely white box". A "relatively black box" can be recognized via mathematical transformations [1].

19. In the switch-technology, a system of interconnected transition graphs is used as the algorithm model to support the possibility of composition and decomposition of algorithms and ensure practical application in designing complex logic control systems. Moreover, automata (transition graphs) can generate centralized, decentralized, and hierarchical structures [1].

20. If a system contains N transition graphs with arbitrary number of states, then only N internal multivalued variables of the state coding scheme can be used in programs even with regard for the interaction of graphs.

Both in automaton and object-oriented programming, automata exchange messages. As a result of the possibility of interaction between automata constituting a program via exchange of the numbers of their internal states, the

automaton programming differs from the object-oriented programming, in which objects are regarded as "black boxes" with encapsulated internal contents.

21. Transition graphs of a system may interact according to the "inquiry-answer-deletion" or "inquiry-answer" principle by exchanging the numbers of states.

22. Besides the exchange of the numbers of states by automata generated by sequential switches in a program, transition graphs may also interact according to the embedding principle. This can be implemented by embedded switches of arbitrary depth or functions constructed from these switches that realize transition graphs of a specified embedding depth.

23. These graph-graph interaction principles support hierarchical algorithm design. In automaton technology, "up-down" and "down-up" algorithm designs are possible. In the "up-down" strategy, it is easy to design correct algorithms. For example, if the algorithm is not state-parallel in design, every vertex in the initial transition graph can be replaced by a fragment containing a few vertices, other transition graphs can be nested in this fragment, and other transition graphs can be accessed from this fragment. Every vertex in the structure thus obtained, in turn, can be refined. The program realizing this algorithm must be isomorphic to the algorithmic structure.

24. For an algorithm designed as a system of transition graphs, this system, if possible, must be decomposed into disconnected subsystems formed by interconnected transition graphs and a graph of attainable labels (states) describing the functionalities is constructed for each subsystem, if dimension permits.

25. A control automaton is defined as a set of an automaton and functional delay elements. These elements are regarded as a controlled object (servos and signalizers): along with the values of "object" output variables, the automaton also generates the values of "time" output variables. Along with the values of "object" input variables, the automaton receives the values of "time" input variables. Both "time" and "object" variables are shown on transition graphs. Different approaches to program realization of functional delay elements by functions are examined in [1].

26. Transition graphs can also be used in designing models for logic control objects and describing and modeling both open and closed "control algorithm-model of controlled object" complexes from a unified standpoint.

27. In the automaton technology, the control program specifications must consist of a closed "data source-system of interconnected automata-functional delay elements-servos-data representation media" connection scheme describing the interfaces of automata and a system of interconnected transition graphs describing the behavior of automata in the scheme. The connection circuit (some analog of the data ow diagram used in structural system analysis) and transition graphs must be displayed compactly (whenever possible) to aid in understanding the specifications (the Gestalt description requirements).

28. The switch-technology permits only one language for algorithm (transition graph) specifications under different programming languages. Methods of realization of formal and isomorphic transition graphs by programs in different languages used in control devices, including PL controllers, are described in [1].

29. The automaton approach is helpful to the customer, designer, technologist, user, and controller in understanding what has been done, what is being done, and what must be done in the program project. It is useful in distributing the work and, most importantly, the responsibility between different specialists and organizations. Such a division of labor is pivotal particularly in projects with foreign participation due to language barriers and misunderstanding. The approach is helpful in handling project details even at the early design stage and in demonstrating them to the customer in a convenient form.

30. This technology permits communication between designers in terms of technological processes in a formalized and understandable language (a sort of technical Esperanto in which they can communicate, for example, as follows: "change the value from zero to one at the fourth position at the fifth vertex in the third transition graph") to avoid misunderstanding due to confusion even within one language and when participants of different countries are involved in a project and no specialist knowledgeable in the technological process is required to introduce changes in the program.

31. In this approach, the programmer need not know the technological process and the designer need not know the details of programming. The application programmer need not do the jobs of the customer, technologist, and designer, but restrict himself to the realization of formalized specifications. Thus, the fields of knowledge he must possess can be narrowed and ultimately his job can be automated. The "traces" of work of one designer can be retained so that program support and modification can be done by some other specialists. It is also effective in controlling the design and text of programs, not just the results as in most cases.

32. In implementing transition graphs, the values of "time" output variables are replaced by functions realizing functional delay elements and the values of "object" output

variables can be replaced by functions of other types. Therefore, this approach can also be applied in implementing the control part of logic algorithms, as demonstrated by the program compiled for synchronizing the generator with the main distribution board bus [1].

33. The automaton technology was successfully applied in designing control systems for ship equipment and other projects based on diverse computing apparatus employing different programming languages.

34. Design, testing, and maintenance of several logic control systems have corroborated the effectiveness of switch-technology, at least, for the systems under review. According to Norcontrol (Norway), this approach produced a high quality logic control system for the Marine diesel generator DGR-2A 500 * 500. Its application to other control problems is described in [1].

35. Thus, this technology is very effective for algorithmization and programming.

36. This technology can be characterized by seven parameters: state, automaton, prehistory-independence, multivalued state coding, a system of interconnected transition graphs, formal and isomorphic programming, and switch constructs. The formal and isomorphic programs constructed from transition graphs by this technology are demonstrative, comprehensible, structurable, addressable, embeddable, hierarchical, controllable, observable and reliable.

37. In other problems, the states of the control automaton described by a transition graph must be distinguished from the memory states. While the number of automaton states is usually not greater a few tens, the number of memory states is far greater than this amount. Therefore, they are not identified explicitly. If the states are not partitioned in this manner, states are not used and the program behavior is determined as a set of actions in response to events without reference to the automaton states with which these events are associated. Furthermore, the control variables must be distinguished from the internal variables of other attributes.

38. In this technology, control automata can be designed for individual modes (gang valve opener and closer), combined modes (close-open automaton for a valve gang), and individual objects (valves) for implementing individual or combined modes.

39. If the initial algorithm is realized by a single-input single-output graph scheme, it can be applied jointly with other methods to construct the transition graph of the automaton [1] realized by a switch construct and "do…while" operator with a condition defined by the number of the terminal vertex. This method is more effective than the Ashcroft and Mann method [1].

40. In nonprocedural specification of automata as transition and output tables, rows and columns (or vice versa) define states and events. Therefore, processing sequence (state-event or event- state) for the interpreter in nonprocedural realization of automata is immaterial [1]. In procedural specification of automata as algorithmic or program schemes, realization is procedural, requires less memory, and depends on the state and event processing sequence.

41. This technology is complete and transparent, because it embraces all stages of algorithmization and programming for which methods guaranteeing high-quality for the design of a project as a whole are available.

42. A detailed description of this approach is given in [1-6].

3. Evolution and Applications of the Approach

Switch-technology for the logical control tasks was extended for the "reactive" systems, controlled by events [3]. Then we developed object-oriented programming with obvious state extracting [6] (automata as classes; formal method of conversion class-oriented automata programs to a program in "classical switch-technology" (it allows to develop model of system using PC and then formally translate it into the microcontrollers platform); automata as member functions; state patterns; we also showed that automata have properties of objects, used in object-oriented paradigm). Switch-technology can be used also for realization of calculating algorithms.

In our method an automaton has three applications: for specification, implementation and tracing (protocoling).

On http://is.ifmo.ru there are a lot of complete examples of the controlling systems and other tasks, which illustrates the offered approach.

4. Conclusions

General advantage of this technology is in developing of program documentation simultaneously with the implementation. So we declared a new course in programming: Course for Open Program Documentation, – as a logical progress of Open Source course. The word "documentation" means here the union of a traditional documentation, commented source code and charts and diagrams (state transition graphs and others), which were used for developing. This trinity is exhaustive for understanding, and further developing of the system from any stage of its elaboration.

Switch-technology is very useful and powerful approach for multi-agent systems engineering, developing and documenting.

REFERENCES

[1] Shalyto, A., *SWITCH-Technology. Algorithmic and Programming Methods for Solution of the Logic Control Problems*. Nauka (Science), St.Petersburg (1998)

[2] Shalyto, A., Logic Control and "Reactive" Systems Algorithmization and Programming // *Automation and Remote Control*, Vol. 62. N1. (2001)

[3] Shalyto, A., Tukkel, N., SWITCH-Technology, An Automated Approach to Developing Software for Reactive Systems // *Programming and Computer Software*, 27(5) (2001)

[4] Shalyto, A., Logic Control. *Hardware and Software Algorithms Implementation*. Nauka (Science), St.Petersburg (2000)

[5] Shalyto, A., Software Automation Design, Algorithmization and Programming of Problems of Logical Control // *Journal of Computer and Systems Sciences International*, vol. 39. N6 (2000)

[6] Shalyto, A., Tukkel, N., Control System for the Tank for Robocode Game. Project documentation. – http,//is.ifmo.ru

[7] Shalyto, A., Tukkel, N., Translating Iterative Algorithms into Automation Once // *Programming and Computer Software*, 28(5) (2002)

[8] Shalyto, A., Cognitive Properties of Hierarchical Representations of Complex Logical Structures // *Architectures for Semiotic Modeling and Situation Analysis in Large Complex Systems. Proceedings of the 1995 International Symposium on Intelligent Control. Monterey, California* (1995)

[9] Deviatkov, V., *Systems of Artificial Intelligence*. MSTU (Moscow State Technical University) of N. Bauman Publishers, Moscow (2001)

[10] Gavrilova, T., Horoshevsky, V., *Knowledge Bases of Intellectual Systems*. Piter, St.Petersburg (2000)