The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

SEQUENT MODEL FOR REPRESENTATION OF DIGITAL SYSTEMS BEHAVIOR

Arkadij ZAKREVSKIJ

Institute of Engineering Cybernetics NAS B, Surganov Str. 6, 220012, Minsk, Belarus
e-mail: *zakr@newman.bas-net.by*

Abstract. *A model of sequent automaton is proposed for description of digital systems behavior in the space of Boolean variables: input, output and inner ones. The rules of its equivalence transformations are formulated, leading to several canonical forms. Simple sequent automaton is introduced, represented in matrix form, which is intended for easing PLA implementation of the automaton. The problem of automata correctness is discussed.*

Key Words. *Digital system behavior, Boolean space of events, sequent automaton, canonical forms, checking for correctness, PLA implementation*

1. EVENTS IN THE BOOLEAN SPACE

Many complex engineering systems may be regarded as dynamic digital systems working in some surroundings. Very often their behavior can be expressed in terms of Boolean variables taking their values from the set $\{0, 1\}$ and defining in such a way the states of individual elements of the system. Usually, when a control system is constructed to ensure the proper interaction between system components, the set W of all variables is divided into three classes: X, Y and Z. Input variables (X) present information received from sensors situated in surroundings or in the system itself; output variables (Y), calculated inside the system, are intended for control purposes and used by executing elements; and inner variables (Z) may play both roles and could be considered as memory of the system.

$2^{|W|}$ different combinations of values of variables from W constitute the Boolean space over W ($|W|$ denotes the cardinality of set W). This Boolean space is designated below as BS(W). Each of its elements may be regarded as a global state of the system, or as the corresponding event that occurs when the system enters that state. Let us call such an event *elementary*. In the same way, the elements of Boolean spaces over X, Y and Z may be regarded as input states, output states and inner states, as well as corresponding events.

Besides that far more events of other types may be introduced into consideration. Generally, every subset of BS(W) may be interpreted as an event which occurs when some element from BS(W) is realized, i. e. when the variables from W take the corresponding combination of values. In this general case the event is called *complicated* and could be presented by the characteristic Boolean function of the regarded subset. So, the number of complicated events coincides with the number of arbitrary Boolean functions of $|W|$ variables.

From the practical point of view, the following two types of events deserve special consideration: basic events and simple events.

Basic events are represented by literals - symbols of variables or their negations - and occur when these variables take corresponding values. For example, basic event a occurs when variable a equals 1, and event c' occurs when $c = 0$. The number of different basic events is $2|W|$.

Simple events are represented by elementary conjunctions, and occur when these conjunctions take value 1. For example, event $ab'f$ occurs when $a = 1$, $b = 0$ and $f = 1$. The number of different simple events is $3^{|W|}$, including trivial event, when values of all variables are arbitrary.

Evidently, the class of simple events absorbs elementary events and basic events. So, elementary conjunction k_i is the general form for representation of events i of all three introduced types; it contains symbols of all variables in the case of an elementary event and only one symbol when a basic event is regarded. One event i can realize another j - it means that the latter always comes when the former comes. It follows from the definitions, that it occurs when conjunction k_i implicates conjunction k_j , in other words, when k_j can be obtained from k_i by deleting some of its letters. For example, event $abc'de'$ realizes events $ac'd$ and $bc'e'$, event $ac'd$ realizes basic events a, c' and d, etc. Hence, several different events can occur simultaneously, if only they are not orthogonal.

2. SEQUENT AUTOMATON

The behavior of a digital system is defined by the rules of changing its state. A standard form for describing such rules was suggested by the well-developed classical theory of finite automata considering relations between the sets of input, inner and output states. Unluckily, that model becomes inapplicable for digital systems with many Boolean variables - hundreds and more. That is why a new formal model was proposed in [3-5] called sequent automaton. It takes into account the fact, that interaction between variables from W takes place within comparatively small groups and has functional character. And it suggests means for describing both the control unit of the system and the object of control - the body of the system.

Sequent automaton is a logical dynamic model defined formally as a system S of *sequents* s_i. Each sequent s_i has the form $f_i \,|\!-\, k_i$ and defines the "cause-effect" relation between some complicated event represented by Boolean function f_i and a simple event k_i represented by conjunction term k_i ; |- is the symbol of the considered relation. Suppose function f_i is given in disjunctive normal form (DNF).

The expression $f_i \,|\!-\, k_i$ is interpreted as follows: if at some moment function f_i takes value 1, then immediately after that k_i must also become equal to 1 - by that the values of all variables in k_i are defined uniquely. In such a way a separate sequent can present a definite demand to the behavior of the discrete system, and the set S as a whole - the totality of such demands.

Note, that the variables from X may participate only in f_i and can carry information got from some sensors, the variables from Y present control signals and participate only in k_i, and the variables from Z are feed-back variables which can be presented both in f_i and k_i.

The explication of "immediately after that" depends greatly on the accepted time model. It is different for two kinds of behavior interpretation, which could be used for sequent automata, both of practical interest: synchronous and asynchronous.

We shall interpret system S mostly as a *synchronous* sequent automaton. In this case the behavior of the automaton is regarded in discrete time t - the sequence of moments $t_o, t_1, t_2, ..., t_l, t_{l+1}, ...$. At a current transition from t_l to t_{l+1} there are executed simultaneously all such sequents s_i for which $f_i = 1$ and as a result all corresponding conjunctions k_i turn to 1 (all their factors take value 1). In that case "immediately after that" means "at the next moment".

Suppose that if some of inner and output variables are absent in conjunctions k_i of executed sequents, they preserve their previous values. That is why the regarded sequent automata are called *inertial* [4]. Hence a new state of the sequent automaton (the set of values of inner variables) is defined uniquely, as well as new values of output variables.

Sometimes the initial state of the automaton is fixed (for moment t_o), then the automaton is called *initialized*. The initial state uniquely determines the set R of all reachable states. When computing it, it is supposed that all input variables are free, i. e. by any moment t_l they could take arbitrary combinations of values. Let us represent set R by characteristic Boolean function φ of inner variables which takes value 1 on the elements from R. In the case of non-initialized automata it is reasonable to consider that φ = 1.

Under *asynchronous* interpretation the behavior of sequent automaton is regarded in continuous time. There appear a lot of more hard problems of their analysis connected with races between variables presented in terms k_i, especially when providing the automaton with important quality of correctness.

3. EQUIVALENCE TRANSFORMATIONS AND CANONICAL FORMS

Let us say that sequent s_i is *satisfied* in some engineering system if event f_i is always followed by event k_i. And sequent s_i *realizes* sequent s_j if the latter is satisfied automatically when the former is satisfied.

Affirmation 1. Sequent s_i realizes sequent s_j if and only if $f_j \Rightarrow f_i$ and $k_i \Rightarrow k_j$, where $\Rightarrow$ is the symbol of formal implication.

For instance, sequent $ab \vee c$ |- uv' realizes sequent abc |- u. Indeed, $abc \Rightarrow ab \vee c$ and $uv' \Rightarrow u$.

If two sequents s_i and s_j realize each other, they are *equivalent*. In that case $f_i = f_j$ and $k_i = k_j$.

The relations of realization and equivalence can be extended onto sequent automata S and T. If S includes in some form all demands contained in T, S realizes T. If two automata realize each other, they are equivalent.

These relations are easily defined for *elementary sequent automata* S^e and T^e, which consist of *elementary sequents*. Left part of such a sequent presents an elementary event in BS($X \cup Z$), right part presents a basic event (for example, $ab'cde'$ |- q, where it is supposed that $X \cup Z = \{a, b, c, d, e\}$. S^e realizes T^e if it contains all sequents contained in T^e. S^e and T^e are equivalent if they contain the same sequents. It follows from here that elementary sequent automaton is a *canonical form*.

There exist two basic equivalencies formulated as follows.

Affirmation 2. Sequent $f_i \vee f_j$ |- k is equivalent to the pair of sequents f_i |- k and f_j |- k.

Affirmation 3. Sequent f |- $k_i k_j$ is equivalent to the pair of sequents f |- k_i and f |- k_j.

According to these affirmations any sequent can be decomposed into a series of elementary sequents (which cannot be decomposed further). That transformation enables to compare any sequent automata checking them for binary relations of realization and equivalence.

Affirmations 2 and 3 can be used for equivalence transformations of sequent automata by elementary operations of two kinds: splitting sequents (changing one sequent for a pair) and merging sequents (changing a pair of sequents for one, if possible).

Elementary sequent automaton is useful for theoretical constructions but could turn out quite non-economical when regarding some real control systems. Therefore two more canonical forms are introduced.

The *point sequent automaton* S^p consists of sequents in which all left parts represent elementary events (in BS($X \cup Z$)) and are different. The corresponding right parts show the responses. This form can be obtained from elementary sequent automaton S^e by merging sequents with equal left parts.

The *functional sequent automaton* S^f consists of sequents in which all right parts represent basic events in BS($Z \cup Y$) and are different. So the sequents have the form $f_i^1 \,|\!-\, u_i$ or $f_i^0 \,|\!-\, u_i'$, where variables $u_i \in Z \cup Y$, and the corresponding left parts are interpreted as switching functions for them: on-functions f_i^1 and off-functions f_i^0. S^f can be obtained from S^e by merging sequents with equal right parts.

Note that both forms S^p and S^f can be obtained also from arbitrary sequent automata by disjunctive decomposition of the left parts of the sequents (for the point sequent automaton) or conjunctive decomposition of the right parts (for functional one).

4. SIMPLE SEQUENT AUTOMATON

Consider now a special important type of sequent automata - a *simple sequent automaton.* It is defined formally as a system S of *simple sequents* - expressions $k_i' \,|\!-\, k_i''$ where both k_i' and k_i'' are elementary conjunctions representing simple events. This form has a convenient matrix representation, inasmuch as every elementary conjunction can be presented as a ternary vector.

Let us represent any simple sequent automaton by two ternary matrices: a *cause matrix* $\boldsymbol{A}$ and an *effect matrix* $\boldsymbol{B}$. They have equal number of rows indicating simple sequents, and their columns correspond to Boolean variables - input, output and inner ones.

Example. The two ternary matrices

$$\boldsymbol{A} = \begin{array}{cccccc} a & b & c & p & q & r \\ 1 & - & - & - & 0 & - \\ - & 0 & 1 & 1 & - & - \\ 0 & 1 & - & - & 1 & 1 \\ - & - & 0 & - & - & 0 \\ - & - & 0 & 1 & 0 & - \end{array}, \quad \boldsymbol{B} = \begin{array}{ccccccc} p & q & r & u & v & w & z \\ - & 1 & - & - & 1 & - & 1 \\ - & - & 0 & 1 & - & 0 & - \\ 1 & 0 & - & - & 1 & - & 0 \\ 0 & - & - & - & - & 1 & - \\ - & 1 & 1 & 0 & - & 1 & - \end{array}$$

represent the following system of simple sequents regarded as a simple sequent automaton:

$$\begin{array}{c} aq' \,|\!-\, qvz\,, \\ b'cp \,|\!-\, r'uw'\,, \\ a'bqr \,|\!-\, pq'vz'\,, \\ c'r' \,|\!-\, p'w\,, \\ c'pq' \,|\!-\, qru'w\,. \end{array}$$

Here $X = \{a, b, c\}$, $Y = \{u, v, w, z\}$, $Z = \{p, q, r\}$.

It has been noted [1] that, to a certain extent, simple sequents resemble the sequents of the theory of logical inference, which were introduced by Gentzen [2]. The latter ones are defined as expressions

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$$

that connect arbitrary logic formulae $A_1, \ldots, A_n, B_1, \ldots, B_m$ and are interpreted as implications

$$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m .$$

The main difference is that any simple sequent $k_i' \mid\!- k_i''$ presents not pure logical but cause-effect relation: event k_i'' is generated by event k_i' and appears after it, so we cannot mix variables from k_i' with variables from k_i''.

But sometimes we may discard this time aspect and consider terms k_i' and k_i'' on the same level, for instance, when looking for stable states of the regarded system. In that case sequent $k_i' \mid\!- k_i''$ could be formally changed for implication $k_i' \rightarrow k_i''$ and subjected further to Boolean transformations, leading to equivalent sets of Gentzen sequents and corresponding sets of standard disjuncts usual for theory of logical inference.

For example, in such a way the system of simple sequents

$$ab \mid\!- cd', \quad a'b' \mid\!- cd, \quad a'b \mid\!- c$$

may be transformed into the following system of disjuncts:

$$a \vee b \vee d, \quad a' \vee b' \vee d', \quad a' \vee c' \vee d, \quad b \vee c' \vee d'.$$

5. APPLICATION IN LOGIC DESIGN

The model of simple sequent automaton is rather close to the well-known technique of disjunctive normal forms (DNF) used for hardware implementation of systems of Boolean functions [6]. Indeed, each row of matrix $\boldsymbol{A}$ may be regarded as a conjunctive term (product), and each column in $\boldsymbol{B}$ defines DNFs for two switching functions of the corresponding output or inner variable: 1s indicate terms entering ON-functions, while 0s indicate terms which enter OFF-functions. Note, that these DNFs can be easily obtained by transforming the regarded automaton into S^f-form and changing after that expressions $f_i^1 \mid\!- u_i$ for $u_i^1 = f_i^1$ and $f_i^0 \mid\!- u_i'$ for $u_i^0 = f_i^0$. For the same example

$p^1 = a'bqr$, $p^0 = c'r'$; $\quad q^1 = aq' \vee c'pq'$, $q^0 = a'bqr$; $\quad r^1 = c'pq'$, $r^0 = b'cp$;
$u^1 = b'cp$, $u^0 = c'pq'$; $\quad v^1 = aq' \vee a'bqr$; $\quad w^1 = c'r' \vee c'pq'$, $w^0 = b'cp$;
$z^1 = aq'$, $\quad z^0 = a'bqr$.

It is seen from here that the problem of constructing a simple sequent automaton with minimum number of rows is similar to the minimization of a system of Boolean functions in the class of DNFs known as a hard combinatorial problem. An approach to its solving was suggested in [7, 8].

The considered model turned out to be especially convenient for representation of programmable logic arrays (PLA) with memory on RS-flip-flops. It is used also in methods of automaton implementation of parallel algorithms for logical control described by expressions in PRALU [11].

Consider a simple sequent automaton shown in the above example. It is implemented by a PLA represented below. It has three inputs (a, b, c) supplied with inverters (NOT-elements) and four outputs (u, v, w, z) supplied with RS-flip-flops. So its input and output lines are

doubled. The six input lines are intersecting with five inner ones, and at some points of intersection transistors are placed. Their disposition can be presented by a Boolean matrix easily obtained from matrix ***A***, and determines the AND-stage of the PLA. In a similar way the OR-stage of the PLA is found from matrix ***B*** and realized on the intersection of inner lines with 14 output ones.

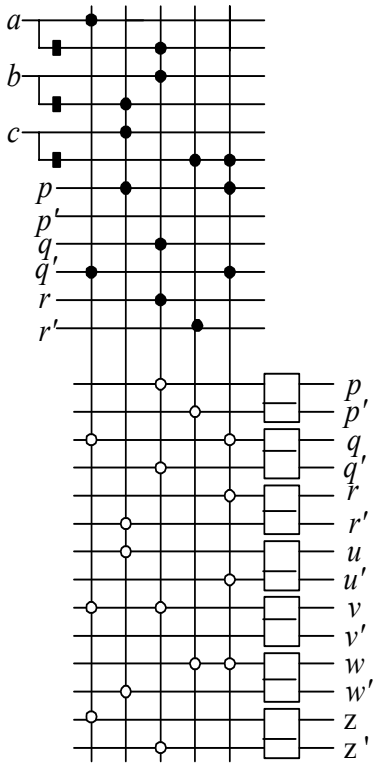

Figure 1. PLA implementation of a sequent automaton

6. CHECKING FOR CORRECTNESS

In general, correctness is a quality of objects of some type, defined as the sum of several properties, which are considered reasonable and necessary [10].

Let us enumerate such properties first for synchronous sequent automata.

Evidently, for any sequent s_i which carries some information inequalities $f_i \neq 0$ and $k_i \neq 1$ should hold, to avoid trivial sequents.

Sequents s_i and s_j are called *parallel* if they could be executed simultaneously. A necessary and sufficient condition of parallelism for non-initialized automaton is relation $f_i \wedge f_j \neq 0$, for initialized - relation $f_i \wedge f_j \wedge \varphi \neq 0$.

First of all, any sequent automaton should be *consistent*, that is very important. That means that for any parallel sequents s_i and s_j relation $k_i \wedge k_j \neq 0$ must hold. Evidently, this condition is necessary, inasmuch as by its violation there exists some variable that must take two different values which is impossible.

The second quality is not so necessary for sequent automata as the first one, but is also useful. It is *irredundancy*. A system S is *irredundant* if it is impossible to delete from it some sequent or if only a literal from a sequent without violating the functional properties of the system. For example, it should not have "non-reachable" sequents, such s_i for which $f_i \wedge \varphi = 0$.

It is rather easy to check a simple sequent automaton for consistency. An automaton represented by ternary matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ is obviously consistent if for any orthogonal rows in matrix $\boldsymbol{B}$ the corresponding rows of matrix $\boldsymbol{A}$ are also orthogonal. Note that this condition is satisfied in Example.

One more useful quality called *persistency* is very important for asynchronous sequent automata. To check them for this quality it is convenient to deal with the functional canonical form.

The point is that several sequents can be executed simultaneously and if the sequent automaton is asynchronous, these sequents (called parallel) could compete - the so called *race* could take place. The automaton is *persistent* if the execution of one of the parallel sequents does not destroy the conditions for executing other ones.

Affirmation 4. In a persistent asynchronous sequent automaton for any pair of parallel sequents

$$f_i^1 \text{ |- } u_i \text{ and } f_j^1 \text{ |- } u_j ,$$
$$f_i^0 \text{ |- } u_i' \text{ and } f_j^1 \text{ |- } u_j ,$$
$$f_i^1 \text{ |- } u_i \text{ and } f_j^0 \text{ |- } u_j' ,$$
$$f_i^0 \text{ |- } u_i' \text{ and } f_j^0 \text{ |- } u_j'$$

the corresponding relation should hold:

$$f_i^1 f_j^1 : u_i' u_j' \Rightarrow (f_i^1 : u_i' u_j)\,(f_j^1 : u_i u_j'),$$
$$f_i^0 f_j^1 : u_i u_j' \Rightarrow (f_i^0 : u_i u_j)\,(f_j^1 : u_i' u_j'),$$
$$f_i^1 f_j^0 : u_i' u_j \Rightarrow (f_i^1 : u_i' u_j')\,(f_j^0 : u_i u_j),$$
$$f_i^0 f_j^0 : u_i u_j \Rightarrow (f_i^0 : u_i u_j')\,(f_j^0 : u_i' u_j),$$

where expression $f : k$ means the result of substitution those variables of function f which enter elementary conjunction k for the values satisfying equation $k = 1$.

The proof of this affirmation can be found in [9].

ACKNOWLEDGMENT

This research was supported by ISTC, Project B-104-98.

REFERENCES

[1] M.Adamski, *Digital systems design by means of rigorous and structural method.* - Zielona Gora, 1990 (in Polish).

[2] G.Gentzen, "Untersuchungen über das logische Schließen", *Mat. Z., v. 39* (1934-35), pp. 176-210, 405-431.

[3] V.S.Grigoryev, A.D.Zakrevskij, V.A.Perchuk, "The sequent model of the discrete automaton",. *Vychislitelnaya tekhnika v mashinostroenii,* March 1972, Minsk, Institute of Engineering Cybernetics, pp.147-153 (in Russian).

[4] V.N.Zakharov, "Sequent description of control automata", *Izvestiya AN SSSR,* 1972, №2 (in Russian).

[5] V.N.Zakharov, *Automata with distributed memory.* Moscow: Energia, 1975 (in Russian).

[6] A.D.Zakrevskij, V.S.Grigoryev, “A system for synthesis of sequent automata in the basis of arbitrary DNFs”, *Problems of Cybernetics. Theory of relay devices and finite automata*. VINITI, Moscow, 1975, pp. 157-166 (in Russian).
[7] A.D.Zakrevskij, “Optimizing sequent automata”, *Optimization in digital devices design,* L., 1976, pp. 42-52 (in Russian).
[8] A.D.Zakrevskij, “Optimizing transformations of sequent automata”, *Tanul. MTA SeAKJ,* Budapest, 63/1977, p. 147-151 (in Russian).
[9] A.D.Zakrevskij, *Logical synthesis of cascade networks.* Moscow: Nauka, 1981 (in Russian).
[10] A.D.Zakrevskij, “The analysis of concurrent logic control algorithms”, *Lecture Notes in Computer Science, vol. 278,* Springer Verlag, 1987, pp. 497-500.
[11] A.D.Zakrevskij, *Parallel algorithms for logical control.* Minsk, Institute of Engineering Cybernetics, 1999 (in Russian).