Grzegorz ŁABIAK
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI,
ul. Licealna 9, 65-417 Zielona Góra, Poland

Conception of partial specification of logic controller deterministic state machine

Dr inż. Grzegorz ŁABIAK

Pracuje w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego jako adiunkt. Zainteresowania naukowe koncentrują się wokół metod projektowania układów cyfrowych, technik programowania oraz formalnych metod weryfikacji systemów dyskretnych. Jest autorem i współautorem licznych publikacji o zasięgu krajowym i międzynarodowym.

e-mail: G.Labiak@iie.uz.zgora.pl

Abstract

The issue of creating a deterministic behaviour description (without conflicts between transitions) of a logic controller is an issue of computational complexity equal to a classic satisfiability problem. Its solution through by-hand transition predicates construction can be very painstaking and tedious, if possible at all. The paper focuses on automatic transition predicates construction where transition predicates are partially specified by a designer giving only the most essential information necessary to comprehend the controller behaviour. This partial specification causes that a state machine (statechart diagram) is nondeterministic (and hence cannot be technically implemented) and the CAD system task is to transform automatically this partial specification into a corresponding deterministic form.

Keywords: deterministic state machine, statechart diagrams, logic controller, transition conflicts, predicates, Boolean algebra, symbolic methods.

Koncepcja niepełnej specyfikacji deterministycznej maszyny stanów sterownika logicznego

Streszczenie

Zagadnienie stworzenie deterministycznego opisu zachowania sterownika logicznego (bez konfliktów między tranzycjami) opisanego diagramami statecharts [2, 9] jest zagadnieniem o złożoności problemu spełnialności i jego rozwiązanie poprzez „ręczne" konstrukcje predykatów tranzycji może być dla projektanta bardzo trudne o ile w ogóle możliwe. Referat skupia się na problemie automatycznego doboru predykatów tranzycji w warunkach niepełnej specyfikacji, tak aby maszyna stanów [2, 9] (diagram statechart, rys. 3a) modelowała zachowanie w sposób deterministyczny. W proponowanej metodzie projektant podaje najistotniejsze zdarzenia warunkujące przejścia, a system *CAD* automatycznie rozwiązuje konflikty tranzycji niepełnej specyfikacji. Podstawową koncepcją automatycznego rozwiązywania konfliktów tranzycji jest utworzenie własnego zestawu wszystkich zmiennych dla każdej tranzycji osobno (podrozdz. 4.1). Nowo utworzone zmienne wyznaczają bazę symbolicznej przestrzeni wektorowej, w której można przestawić wszystkie możliwe predykaty (podrozdz. 4.2). W przestrzeni symbolicznej w łatwy sposób można opisać wyrażeniem logicznym zbiór wszystkich predykatów ortogonalnych oraz zbiory odpaleń wynikające z niepełnej specyfikacji (podrozdz. 4.3). Iloczyn w przestrzeni symbolicznej zbiorów predykatów ortogonalnych i zbiorów odpaleń wyznacza zbiór zestawów ortogonalnych predykatów wynikających z częściowej specyfikacji (podrozdz. 4.4). Wszystkie operacja na zbiorach wykonywane są jako logiczne przekształcenia odpowiednich funkcji charakterystycznych, które efektywnie mogą być implementowane za pomocą binarnych diagramów decyzyjnych [7].

Słowa kluczowe: deterministyczna maszyna stanów, diagramy statechart, sterownik logiczny, konflikty tranzycji, predykaty, algebra Boola, metody symboliczne.

1. Introduction

A digital binary controller can be designed as a classic Finite State Machine (FSM) or with use of a mathematical Petri net model (concurrent controller) or as Hierarchical Concurrent automaton (HCFSM). The latter approach combines both former approaches: is state oriented, uses plain concurrency and, additionally, makes that behaviour is ordered hierarchically. In order to implement the controller physically, as an electronic circuit, it is necessary to create a formal and deterministic model of its behaviour. On the one hand this requires giving all the details of its behaviour what for the human can be very painstaking and tedious, if possible. On the other hand not always every tiny aspect of the behaviour is equally important and some of them can be left unspecified. Such a case takes place when the designer must solve conflicts between transitions and where priorities between transitions are irrelevant. The paper presents how to form partially specified nondeterministic transition predicates and how to find automatically their deterministic form.

2. Transition conflicts

In the paper [3] the authors give the following definition of the conflict: "two compound transitions are in conflict if there is some common state that would be excited if any one of them were to be taken". As opposed to the classic Finite State Machine and Petri Nets transition, conflicts can be grouped into three categories [5]: horizontal, vertical and mixed.

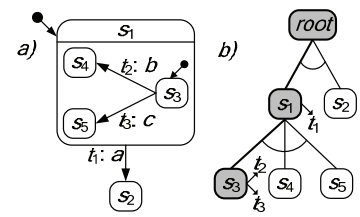

Fig. 1. Transitions in conflict: a) diagram b) hierarchy tree
Rys. 1. Tranzycje w konflikcie: a) diagram b) drzewo hierarchii

The first case (horizontal conflict) takes place when transitions in conflicts are located on the same level of the hierarchy tree. In Fig. 1a transitions t_2 and t_3 are an example of the horizontal conflict and the state that would be excited is state s_3.

The second case (vertical conflict) takes place when conflicting transitions are located on a different hierarchy level (see Fig. 1b). In Fig. 1a the vertically conflicting transition are two sets of transitions $\{t_1,t_2\}$ and $\{t_1,t_3\}$ and their common state in both cases is state s_3 again.

The set of transitions $\{t_1,t_2,t_3\}$ (Fig. 1) combines the features of two former conflicts (horizontal and vertical). Because the transitions in the set are pair wise either vertical or horizontal, this set of transitions is an example of a conflict of the mixed type.

In the hierarchical concurrent automaton (HCFSM), in general, groups of transitions being potentially in conflict make up *structurally inconsistent* set of transitions:

Definition 1. *Structurally inconsistent transitions*

The two transitions t_1 and t_2 are *structurally inconsistent* if $out(t_1) \in hrc^*(out(t_2))$ or $out(t_2) \in hrc^*(out(t_1))$. The set of transitions $\mathbf{T}$ is not structurally consistent (or is inconsistent) if every pair of transitions t_i i $t_j \in \mathbf{T}$ is structurally inconsistent. The set of transitions $\mathbf{T}$ is *maximally inconsistent* ($\mathbf{T}_{I_{\max}}$) if for every transition $t \in \mathbf{T}_z \backslash \mathbf{T}$ ($\mathbf{T}_z$ is the set of all transitions in HCFSM) the set $\mathbf{T} \cup t$ is inconsistent. ■

The function *out*(*t*) gives the transition *t* beginning state, e.g. $out(t_2)=s_3$. The function $hrc^*(s)$ gives the set of states which are hierarchically subordinated to the state *s* together with the state *s*, e.g. $hrc^*(s)=\{s_1, s_3, s_4, s_5\}$.

If the HCFSM controller has to work correctly, the set of maximally inconsistent transitions must have predicates pair wise orthogonal (in the context of HCFSM global state [5]). This means that the transition predicates form *the maximal compatibility class*, where the compatibility relation is the predicates orthogonality relation. For example $\mathbf{T}_{I_{\max_3}} = \{s_1, s_2, s_3\}$. The index *3* in the name of the set $\mathbf{T}_{I_{\max_3}}$ means that transitions belonging to the maximal set of transitions potentially being in conflict take their beginnings from the states which, in the hierarchy tree, are located on the path leading from the root state ($root_z$) to the leaf state s_3 (see. Fig. 1b).

The following set of transition predicates is the example of the predicates which form the maximal compatibility class for the maximally inconsistent transition set $\{t_1,t_2,t_3\}$ for the diagram of Fig. 1:

$$\begin{aligned} t_1 &= a * \overline{b} * \overline{c} \\ t_2 &= b * \overline{a} * \overline{c} \\ t_3 &= c * \overline{a} * \overline{b} \end{aligned} \tag{1}$$

In comparison with the predicates of Fig. 1a the above-mentioned predicates were modified to meet the compatibility relation. The compatibility relation is an orthogonality relation, which means that the predicates pair wise are orthogonal (their product equals zero):

$$\begin{aligned} t_1 * t_2 &= \left(a * \overline{b} * \overline{c}\right) * \left(b * \overline{a} * \overline{c}\right) = 0 \\ t_1 * t_3 &= \left(a * \overline{b} * \overline{c}\right) * \left(c * \overline{a} * \overline{b}\right) = 0 \\ t_2 * t_3 &= \left(b * \overline{a} * \overline{c}\right) * \left(c * \overline{a} * \overline{b}\right) = 0 \end{aligned} \tag{2}$$

and hence the whole diagram of Fig. 1 is the transition conflicts free.

Formally the condition for the HCFSM free from transitions in conflict is as follows [5]:

Definition 2. *HCFSM free from transitions in conflict*

HCFSM is free from transitions in conflict if the following condition is met:

$$\bigvee_{T_{I_{\max}} \subseteq T_z} \; \bigvee_{t_j, t_k \subseteq T_{I_{\max}}} \chi_z * t_k * t_j = 0 \tag{3}$$

where t_j, t_k represent transition predicates and χ_z is a characteristic function of the set of global states of the statechart *Z*. ■

In the case when the event broadcast mechanism is not applied, the context of the global states (χ_z) can be omitted, as it is in the examples in the paper.

3. Partial specification

The basic issue of solving conflicts between transitions is such transition predicates modification that would meet the condition from definition 2. For the diagrams relatively simple, for instance from Fig. 1a, creation of proper predicates is not so difficult, but for the diagrams more complex by-hand creation of orthogonal predicates becomes extremely difficult, if possible at all. The difficulties stem from the fact that the number of sets of maximally inconsistent transitions ($\mathbf{T}_{I_{\max}}$), and all the Boolean expressions (see definition 2), grow in a linear way dependent on the number of simple states in the diagram (e.g. for the diagram from Fig. 1a simple states are $\{s_2, s_3, s_4, s_5\}$). Next, for every expression it must be checked whether the expression equals 0, which is the classic satisfiability problem whose computational complexity is exponential (dependent on the number of logic variables).

The problem of finding all the sets of maximally inconsistent transitions ($\mathbf{T}_{I_{\max}}$) is a classic number of cliques in the graph problem, where vertices are transitions and edges are orthogonality relations. As it was proved in [8] the number of cliques in a graph is:

$$O\left(3^{\frac{m}{3}}\right) \tag{4}$$

where *m* is a number of vertices in a graph, that is, transitions in a diagram. Therefore, it seems that the number of maximally inconsistent transition sets grows exponentially. But in this particular case a linear dependence of the number of maximally inconsistent transition sets on the number of simple states in a diagram is more essential and this implies significant computational simplifications.

Because of the above presented difficulties, it seems to be a good idea to free a designer of specifying full and deterministic (orthogonal) predicates, giving her or him possibility to specify only fundamental (the most essential from designer comprehension point of view) conditions of firing transitions. The predicates specified in this way are only partially specified, making that the behaviour is nondeterministic. Next, basing on the assumed priorities, such partial specification is to be automatically transformed (by means of *CAD* system) into their corresponding deterministic form. The automatic transformation through transition predicates modification, if possible, will make that predicates of potentially conflicting transitions are orthogonal.

Fig. 2a again presents a simple diagram with partially specified predicates of three transitions. Transition t_1 fires when event *a* occurs, transition t_2 fires when event *b* occurs and t_3 fires when event *c* occurs. It is obvious that these three transitions are potentially in conflict and the conflicts take place when at the same instant of time any pair of events or all three events occur. In other words, one can say that a designer did not give any information which one of two or three enabled transitions was to be fired, regarding firing priorities as irrelevant or using firing rules embedded in the *CAD* system. As far as *UML* specification in this respect is concerned [9], only vertical conflicts are solved, passing over other conflicts, namely, lower level hierarchy transitions have priority over higher level transitions.

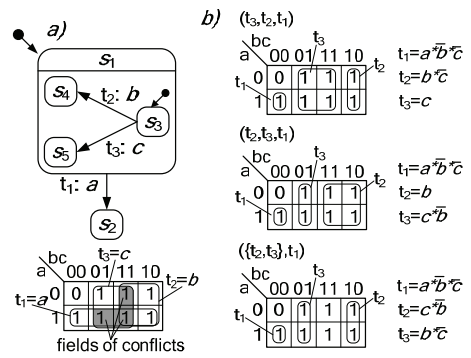

Fig. 2. Partial specification: a) diagram b) three possible deterministic behaviours
Rys. 2. Niepełna specyfikacja: a) diagram b) trzy możliwości deterministycznego zachowania

It seems that in many cases of practical applications the idea of partial predicates specification turns out to be good. But the most essential issue is a question about existence of orthogonal

predicates under given partial specification, and if so, how to find these predicates which meet the assumed priorities.

In order to answer these two questions, the problem of partial specification must be formalized.

The Karnaugh map of Fig. 2a shows that predicates of every transition set the group of its possible firings in the vector space generated on the basis of three variables (vectors) *a*, *b*, *c*. The transition firing in the Karnaugh map is marked as a single minterm within the Karnaugh group, e.g. transition t_1 can fire when the event *a* occurs and events *b* and *c* do not occur, hence the minterm is $a*\overline{b}*\overline{c}$. Common parts of the sets of transition firings related to different transitions (two or three) determine fields of the transition conflicts. Hence, automatic predicate modification should be performed in such a way that:

a) the fields of conflicts should be removed,
b) set of firings (related to predicate transition) before modification should have common part different from the empty set with the set of firings (related do predicate transition) after the modification.

Fig. 2b presents three different predicates modification. In the first case (t_3,t_2,t_1) the highest priority has transition t_3, next t_2 and the lowest priority has transition t_1. This means that at simultaneous occurrence of events *a*, *b* and *c* only transition t_3 related to event *c* will fire. The presence of negated variable *c* in expressions of other two predicates prevents from firing them. In the second case, the priorities (starting from the highest) are as follows (t_2,t_3,t_1). In the third case ($\{t_2,t_3\}$,t_1) transition t_1 has the lowest priority and priorities of transitions $\{t_2,t_3\}$ are equal. This means that simultaneous occurrence of events related to these two transitions, respectively *b* and *c*, makes that none of them will fire. The presented three different prioritizations, in general, are in accordance with the UML [9] guidelines.

Combinatorial comments. In this moment it is worth to direct attention to the fact that the absolute number (independent on partial specification) of all possible sets of orthogonal predicates can be calculated as a sum of the Stirling numbers of the second kind in the following way:

$$L = \underbrace{\left\{ n \atop k \right\}}_{a} + \underbrace{\left\{ n \atop k+1 \right\}}_{b} \tag{5}$$

The Stirling number of the second kind one can interpret as a number of ways of partitioning a set of *n* elements into *k* non empty subsets [1][4]. In this particular case *n* means the number of all possible occurrences of three events (and hence all possible transition firings). This is a permutation with repetition and can be calculated according to the formula:

$$n = 2^{ne} \tag{6}$$

where *ne* is the number of events. Symbol *k* means the number of predicates i.e. number of transitions. Component *a* in Eq. 5 corresponds to a situation when orthogonal predicates (whose number is *k*) cover all possible transition firings. The component *b* corresponds to a situation when orthogonal predicates do not cover all possible transition firings, leaving one non empty set of transition firings (in Fig. 2b uncovered minterms in the Kargnaugh table for all three presented cases). For the diagram from Fig. 2a the absolute number of all possible sets of orthogonal predicates amounts to:

$$L = \left\{ 8 \atop 3 \right\} + \left\{ 8 \atop 3+1 \right\} = 966 + 1701 = 2667 \tag{7}$$

and Fig. 2b presents only three possibilities.

4. The space of deterministic predicates

Under the given partial specification a question of crucial meaning is the question about existence of orthogonal predicates, and if so, how to perform generating the sets of predicates. Generation of the sets of predicates must be performed in such a way that every consecutive set of predicates should be tested whether the given set meets the assumed prioritization. Because predicates are the Boolean expression, it seems to be a good idea to express the sets of predicates also as Boolean equations symbolically coded. Traditionally, the set of Boolean elements can be expressed as a characteristic function, also expressed as the Boolean function, and then every Boolean operation can be efficiently implemented by means of Binary Decision Diagrams [7]. If the set of orthogonal predicates is non empty, this means that the searched predicates exist and finding the proper predicates consists in browsing elements of the set and testing their properties (prioritization).

4.1. Symbolic coding of predicates

The fundamental idea of symbolic coding of predicates is to create for every predicate transition its own copy of full list of events. This is a kind of aliasing the Boolean variables representing events, and as a result *n* events and *k* transitions from the diagram amount to *n*k* new Boolean variables. Fig. 3 presents the simplest fragment of nondeterministic diagram along with two sets of symbolic names of events – the names subscripted with 1 (a_1,b_1) are related to transition t_1 and the names subscripted with 2 (a_2,b_2) are related do transition t_2.

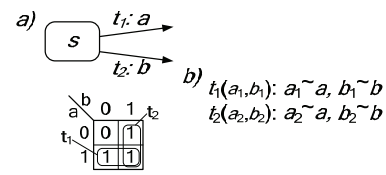

Fig. 3. Variable aliasing: a) diagram b) two sets of symbolic variables
Rys. 3. Aliasowanie zmiennych: a) prosty diagram b) dwa zestawy symbolicznych zmiennych

Newly created variables form the basis of the symbolic predicate vector space. In this space every possible set of predicates ,both orthogonal and nonorthogonal, can be described.

4.2. All orthogonal predicates

In the symbolic predicate vector space it is possible to present the set of all orthogonal predicates. Orthogonal predicates are these whose product equals zero (see def. 2). Fig. 4 presents the Karnaugh table for the diagram from Fig. 2a, where every possible predicate (in new aliased variables) related to transition t_1 labels rows, and predicates related to transition t_2 label columns. The one on the intersection row and column means that the given pair of predicates is orthogonal, zero otherwise.

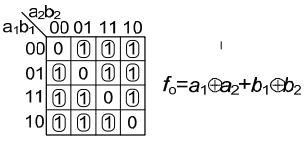

Fig. 4. The set of all orthogonal predicates in the symbolic vector space
Rys. 4. Zbiór wszystkich ortogonalnych predykatów w symbolicznej przestrzeni wektorowej

4.3. The set of transition firings

In the symbolic predicate space among all orthogonal predicates there is also possible to describe the set of all possible transition firings which result from partial specification. The diagram with partial specification is nondeterministic and its predicates are not orthogonal. Fig. 5 on the Karnaugh table for the diagram of Fig. 3a presents groups of possible transition firings resulting from partial specification. For example, transition t_1 can be fired when event a occurs, regardless of the form of predicate t_2, and transition t_2 likewise. The product of the groups of transition firings (shaded in Fig. 5) gives the set of all possible nondeterministic firings, resulting from partial specification.

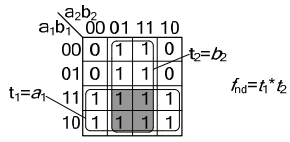

Fig. 5. The sets of firings for partial specification
Rys. 5. Zbiory odpaleń tranzycji dla niepełnej specyfikacji

4.4. The set of orthogonal predicates

The product of all orthogonal predicates and the set of predicates resulting from partial specification gives the set of these orthogonal predicates which meets partial specification. Every element of this set, i.e. every set of predicates, makes that nondeterministic diagram can be transformed into its deterministic form. The only difference between deterministic diagrams is that transition priorities are different. Fig. 6 shows the set of all orthogonal predicates for the diagram from Fig. 3a resulting from partial specification.

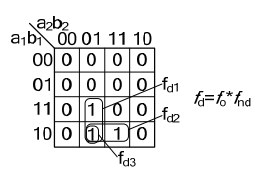

Fig. 6. The set of all orthogonal predicates
Rys. 6. Zbiór wszystkich predykatów ortogonalnych

The set of all orthogonal predicates is described by the following characteristic function:

$$f_d = a_1 b_1 \overline{a}_2 b_2 + a_1 \overline{b}_2 \overline{a}_2 b_2 + a_1 b_1 a_2 b_2 \tag{8}$$

and the set of predicates are implicants of f_d function, e.g.:

$$\begin{aligned} f_{d1} &= a_1 \overline{a}_2 b_2 \quad t_1 = a \quad t_2 = \overline{a} b \\ f_{d2} &= a_1 \overline{b}_1 b_2 \quad t_1 = a\overline{b} \quad t_2 = b \\ f_{d3} &= a_1 \overline{b}_1 \overline{a}_2 b_2 \quad t_1 = a\overline{b} \quad t_2 = \overline{a} b \end{aligned} \tag{9}$$

Every implicant gives the set of orthogonal predicates, in the above presented three implicants there are three possibilities.

5. Summary

The partial specification of the transition predicates frees a designer of description of very many irrelevant details. Partially specified diagrams contain transitions in conflicts and, hence, are nondeterministic. In order to implement them as a digital circuit, it is necessary to solve conflicts automatically. This transformation consists of the following steps:

a) creation of the own sets of all event variable (name aliasing): one set of variable for every transition,

b) computation of the characteristic function of all possible orthogonal predicates in the vector space generated on new variables according to the formula:

$$f_o = \prod_{i=1, j=1}^{k} \sum_{l=1}^{n} x_i^l \oplus x_j^l \quad i \neq j \tag{10}$$

where n is a number of variables and k is a number of transitions,

c) computation of the characteristic function of the set of transition firings (nondeterministic) resulting from partial specification:

$$f_{nd} = \prod_{\forall T_{l\max} \subseteq T_z, \forall t_i \subseteq T_{l\max}} t_i \tag{11}$$

d) computation of the characteristic function of the set of orthogonal predicates resulting from partial specification:

$$f_d = f_o * f_{nd} \tag{12}$$

e) traversal of the set of all orthogonal predicates resulting from partial specification intended for finding the set of predicates with the assumed prioritization,

f) changing the predicates from the symbolic predicates space to variables from the original diagram.

6. References

[1] Graham R.L., Knuth D. E., Patashnik O.: Concrete Mathematics: A Foundation for Computer Science, 2nd Edition, Addison-Wesley Professional, 1994.
[2] Harel D.: Statecharts. A Visual Formalism for Complex Systems, Science of Computer Programming 8, 231–274, 1987.
[3] Harel D. and Naamad A.: The STATEMATE Semantics of Statecharts, ACM Trans. Soft. Eng. Method, 1996.
[4] Lipski W.: Kombinatoryka dla programistów, WNT, Warszawa, 1989.
[5] Łabiak G.: Transition Conflicts Detection in Binary Modular Statechart Diagrams, PDS, ss. 192-197, Kraków 2004.
[6] Łabiak G.: Wykorzystanie hierarchicznego modelu współbieżnego automatu w projektowaniu układów cyfrowych, Oficyna Wydawnicza Uniwersytetu Zielonogórskiego 2005.
[7] Minato S.: Binary Decision Diagrams and Applications for VLSI CAD, Kluwer Academic Publishers, Nov. 1996.
[8] Moon J. W. and Moser L.: On cliques in graphs, Israel Journal of Mathematics, 3(1):23–28, 1965.
[9] UML: OMG Unified Modeling Language TM (OMG UML), Superstructure, Version 2.3, Object Management Group, OMG, 2010.

otrzymano / received: 18.10.2011
przyjęto do druku / accepted: 01.12.2011 *artykuł recenzowany / revised paper*