www.czasopisma.pan.pl PAN www.journals.pan.pl
INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 4, PP. 411–418
Manuscript received October 31, 2012; revised Decemer, 2012. DOI: 10.2478/v10177-012-0056-9

The Problems of Transition Predicates Construction in Hierarchical Concurrent Controllers

Grzegorz Łabiak

***Abstract*—The paper presents a problem of a transition predicates construction in hierarchical concurrent state oriented notation. The notation, called statechart diagrams or state machine, serves as a very convenient formalism for specification of a complex behavior of the embedded systems control unit. The controller specified in this way is discrete, deterministic and synchronous system which operates on binary values and can be implemented in programmable devices as a digital circuit. Well designed controller has conflict-free transitions and its concurrent transitions should be independent. In order to meet this requirements transition predicates must be pairwise both orthogonal and non-implicative. Computational complexities of the problems is equal to classic clique problem. The paper also suggests some statecharts syntactic structures solving these problems.**

***Keywords*—Binary control system, statechart diagrams, hierarchy, concurrency, conflicting transitions, Boolean predicates, computational complexity, compatibility classes, clique problem.**

I. Introduction

MOST today's embedded real-time systems, e.g. telephone, automotive or operating systems are reactive systems. This means that these systems permanently interact with external prompts at pace determined by the environment. They immediately react to the event coming from outside world through generated events. The nature of these systems is quite different than traditional transformational systems, where data output are prepared after some period of time spent on computations. Also different are design methods of these systems; while transformational systems are designed by both imperative and predicative languages, in general, behavior of reactive systems are described by state-oriented methodologies. They bear strong resemblance to traditional controllers and like ordinary controllers they can be treated (see Fig. 1). There can be pointed out three design paradigms of reactive systems: sequentiality, concurrency and hierarchy. These three paradigms are formalized in well known models [1]: Finite State Machines (FSM), Petri nets (also called CFSM – Concurrent FSM) and statecharts (also called Hierarchical CFSM or in UML terminology called state machines [2]). Usually controller generates signals to the controlled object (and controlled object responds to controller, Fig. 1) or menages the data transformation in the data path [3]. If controller operates on binary values it is called binary controller and such a binary controller can easily be implemented in programmable devices as a digital circuit.

G. Łabiak is with Computer Engineering & Electronics Department, University of Zielona Góra, Licealna 9, 65-417 Zielona Góra, Poland (e-mail: G.Labiak@iie.uz.zgora.pl).

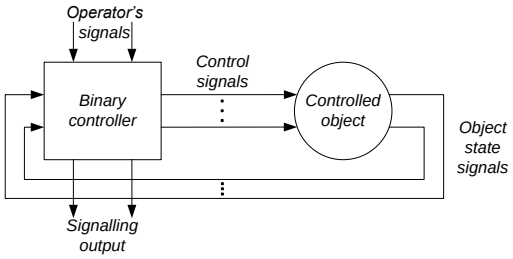


Fig. 1. Binary control system.

II. Syntax and Semantics of HCFSM

Hierarchical concurrent controllers is a controller whose behavior, in principle, can be described in terms of states, concurrency and hierarchy (for example, see Fig. 2). The inventor of this notations, David Harel, called them „Statechart: A Visual Formalism for Complex System" and described them as follows: state + concurrency + hierarchy + broadcast mechanism [4]. The states represent local activities of the system, concurrency means that many states can be active at the same time, and the hierarchy feature allows to assign one superstate many substates. The hierarchy yields hierarchy tree where abstract behaviors are described in terms of more detailed and specialized subbehaviors (see Fig. 3). Broadcast mechanism is a kind of feedback, it means that event generated in one part of the system can affect behavior of the other part of the system. Hierarchy tree is a slightly modified Jackson's diagram [3] where nodes are states (simple, compound and regions), arcs means sequential relationship between states and double arcs means concurrency relationship.

Figure 2 describes behavior of the controller which controls some chemical process [5]. The process consists of measuring out two substrates (superstate *Filling*) and mixing them together in water environment for given period of time (superstate *Process*). Superstate *Initiating* is responsible for preparation of the chemical plant.

III. Sythesis of HCFSM

The issue of hardware synthesis of statecharts is not solved ultimately. There are many implementation schemes depended on target technology. First, published in [6], consists transformation of the statechart into the set of hierarchically linked FSMs traditionally implemented. In [7], a special encoding of the statechart configurations targeted at PLA structures is presented. The drawback of this method is that diagram expresses transitions between simple states only. In [8], Drusinsky enhanced the coding scheme by introducing a prefix-encoding.

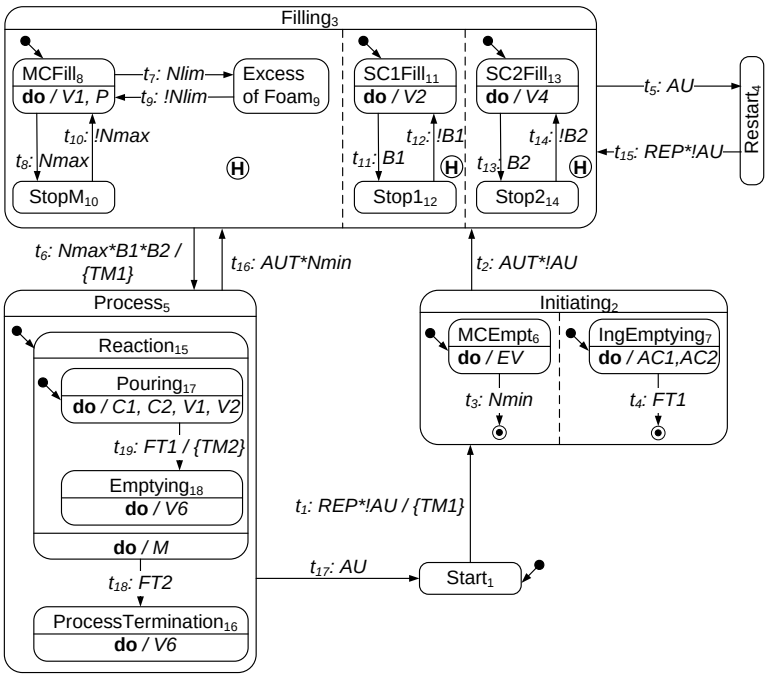


Fig. 2. The example of the reactor's controller.

However, the common drawbacks of the presented methods is the lack of support for history attributes and broadcast mechanism. Other implementation methods using HDL and based on ASIP are presented in [9] and [10], respectively.

To synthesis HSFSM-based logic controller it is necessary to define precisely its behavior in terms of logic values. In Fig. 4 a simply diagram and its waveform illustrate the main dynamic features. Logic value **1** means activity of a state or presence of an event, and value **0** means their absence. When transition t_1 is fired (T = 350) event t_1 is broadcast and becomes available to the system at next instant of discrete time (T = 450). The activity moves from state *START* to state *ACTION*, where entry action (keyword *entry*) and do-activity (ongoing activity, keyword *do*) are performed (events *entr* and *d* are broadcast). Now, transition t_2 becomes enabled. Its source state is active and predicate imposed on it (event t_1) is met. So, at the instant of time T = 450, the system transforms activity to the state *STOP*, performs exit action (keyword *exit*, event ext) and triggers event t_2, which do not affect any other transition. The step is finished.

Summarizing, dynamic characteristics of hardware implementation are as follows:

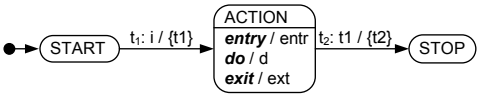
- system is synchronous,
- system reacts to the set of available events through transition executions,
- generated events are accessible to the system during next tick of the clock.

The main assumption of a hardware implementation of HCFSM is that the systems specified in this way can directly be mapped into programmable logic devices. This means that elements from a diagram (e.g. states or events) are in direct correspondence with resources available in a programmable devices — mainly flip-flops and programmable combinatorial logic. Basing on that assumption and taking into account assumed dynamic characteristics, following foundations of hardware implementation can be formulated [11]:

- each state is assigned one flip-flop,
- each event is also assigned one flip-flop,
- based on diagram topography and rules of transition executions, excitation functions are created for each flip-flop in a circuit.

Farther statechart diagrams synthesis description is mainly revolving around specification of flip-flop excitation functions of two type: state flip-flops and event flip-flops. Presented therein the idea of hardware synthesis has been successfully implemented in Author's system called *HiCoS* [12].

a)



b)

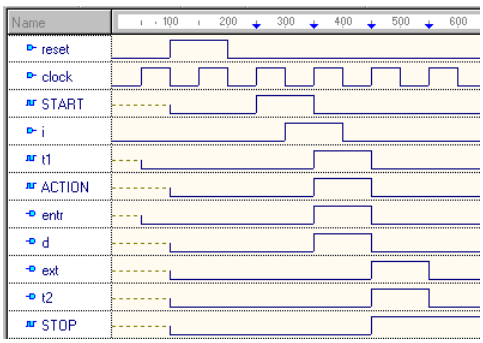


Fig. 4. Simple diagram (a) and its waveform (b).

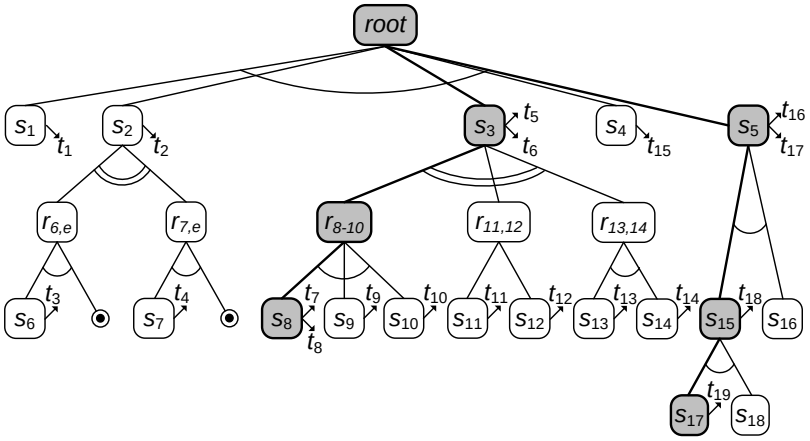


Fig. 3. Hierarchy tree of the reactor.

IV. Well Designed HCFSM

In HCFSM behavior is described mainly by states and transitions. Transition means changing activity in the controller and is executed when the start state is active and logic predicate imposed on it is fulfilled. Logic predicate is a Boolean expression composed of variables which correspond to events. Apart from dynamic properties like liveness and safeness, well designed HCFSM controller should have conflict-free transitions and concurrent transitions should be really independent [13]. The former feature is assured through predicate orthogonality between potentially conflicting transitions and the latter through non-implicativity relation between concurrent transitions.

A. Transition Orthogonality

Two transitions are in conflict if there is some common state that would be exited if any one of them were to be fired [14]. In distinction from FSM and Petri nets statecharts conflicting transitions can be grouped into three categories:

a) horizontal,
b) vertical,
c) mixed.

The first case (a) takes place when the transitions in conflict are on the same level of hierarchy tree. In Fig. 5 horizontally conflicting transitions are t_2 and t_3 and the common state is s_3. The second case (b) holds when conflicting transitions are located on different levels of hierarchy tree. In Fig. 5 vertically conflicting transitions are transitions t_1 and t_2 and also t_1 and t_3, whereas the common state is again s_3. The case of transition t_1, t_2 and t_3 from Fig. 5 at the same time combines features horizontally and vertically conflicting transitions, so the conflict of those three is of mixed type at once.

In HCFSM controllers transitions potentially being in conflict form a set of *structurally inconsistent* transitions.

Definition 1: 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 T is structurally inconsistent if every pair of transitions $t_1, t_2 \in T$ is structurally inconsistent. A set of transitions T is *maximally inconsistent* ($T_{I_{max}}$) if for every transition $t \in T_z \setminus T$ (T_z is a set of all transitions in the HCFSM), the set $T \cup \{t\}$ is consistent (not inconsistent).

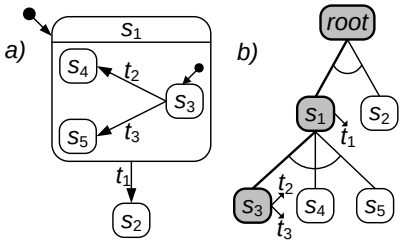


Fig. 5. Potentially conflicting transitions: a) diagram b) hierarchy tree.

The function $out(t)$ gives beginning states of a transition t, e.g. $out(t_2) = s_3$. The function $hrc^*(s)$ gives a set of states which are hierarchically subordinate to s including s, e.g. $hrc^*(s_1) = \{s_1, s_3, s_4, s_5\}$. For HCFSM controllers to work correctly the maximal inconsistent sets of transitions must have predicates pairwise *orthogonal* (in the context of global state [15]), i.e. the transitions predicates form *maximal compatibility classes*, where compatibility means orthogonality relation (e.g. $T_{I_{max_3}} = \{t_1, t_2, t_3\}$). (The index of the set $T_{I_{max_3}}$ is 3 because potentially conflicting transitions stem from the states which on hierarchy tree (Fig. 5b) belong to the path leading from the root to the leaf state s_3.) For example for the diagram from Fig. 5 predicates imposed on transitions could be as follows:

$$t_1 = a * \overline{b} * \overline{c} \quad (1)$$
$$t_2 = b * \overline{a} * \overline{c} \quad (2)$$
$$t_3 = c * \overline{a} * \overline{b} \quad (3)$$

Then the set $T_{I_{max_3}}$ is maximally inconsistent and the predicates of the transitions of $T_{I_{max_3}}$ are pairwise orthogonal:

$$t_1 * t_2 = (a * \overline{b} * \overline{c}) * (b * \overline{a} * \overline{c}) = 0 \quad (4)$$
$$t_1 * t_3 = (a * \overline{b} * \overline{c}) * (c * \overline{a} * \overline{b}) = 0 \quad (5)$$
$$t_2 * t_3 = (b * \overline{a} * \overline{c}) * (c * \overline{a} * \overline{b}) = 0 \quad (6)$$

and hence the diagram is conflict-free.

Formally, the condition for pairwise transition orthogonality of the HCFSM is defined as follows:

Definition 2: The HCFSM is transition-conflict-free when following condition is satisfied:

$$\underbrace{\bigwedge_{T_{I_{max}} \subseteq T_z}}_{a} \underbrace{\bigwedge_{t_j, t_k \in T_{I_{max}}}}_{b} \underbrace{\chi_z * t_k * t_j = \mathbf{0}}_{c} \tag{7}$$

where t_j and t_k represent predicates of respective transitions, $j \neq k$ and χ_z is a characteristic function of the set of global states.

When broadcasting mechanism is not applied (and some other syntactics features) the context of global states (χ_z) can be left out, as it is shown in the example.

B. Transition Non-Implicativity

In case of HCFSM implemented as a digital synchronous automaton two concurrent transitions can potentially fire at the same time and on the other hand they should be independent of each other. The dependencies between two transitions can lead to unintended changes in the controller and to the semantic inconsistencies between controller and controlled object in control system [13] (the state of the controller does not mirror the state of the object, see Fig. 1).

For example transitions in Fig. 6 could have following orthogonal transitions ($T_{I_{max_3}} = \{t_1, t_2\}$, $T_{I_{max_5}} = \{t_1, t_3\}$, $T_{I_{max_6}} = \{t_1, t_4\}$):

$$t_1 = a, \tag{8}$$
$$t_2 = b * \overline{a} \tag{9}$$
$$t_3 = c * \overline{a} \tag{10}$$
$$t_4 = \overline{a} \tag{11}$$

In this diagram firing transition t_2 always entails firing transition t_4 (provided that its start state s_6 is active). This *implicativity relation* between the two transitions stem from the fact that predicates imposed on those two transitions are in implication relation, i.e. satisfying predicate imposed on transition t_2 always makes that predicate imposed on transition t_4 is also satisfied ($t_2 \rightarrow t_4$).

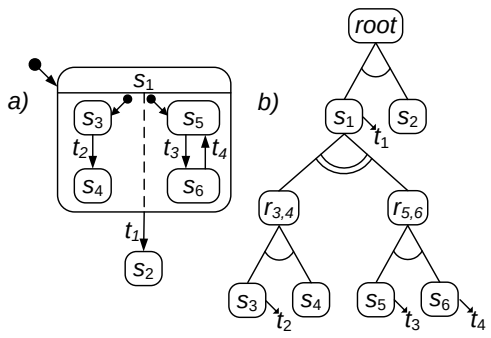


Fig. 6. Dependent concurrent transitions: a) diagram b) hierarchy tree.

In HCFSM concurrent transitions, which can potentially be in implicativity relation, form a set of *structurally consistent* transitions.

Definition 3: [16] Two transitions t_1 and t_2 are *structurally consistent* if $type(lca(\{out(t_1), out(t_2)\})) = AND$. The set of transitions T is structurally consistent if every pair of transitions $t_1, t_2 \in T$ is structurally consistent. A set of transitions T is *maximally consistent* ($T_{C_{max}}$) if for every transition $t \in T_z \setminus T$, the set $T \cup \{t\}$ is not consistent (inconsistent).

The function $lca(S)$, called *lowest common ancestor*, gives a state s which for the set of state S hierarchically covers states belonging to S and there is no other state which covers states from S and is lower in hierarchy then s, e.g. $lca(s_3, s_5) = s_1$. The function $type(s)$ gives type of the compound state s, i.e. if directly subordinate states are in concurrent relationship then the type is AND, otherwise is OR, e.g. $type(s_1) = AND$. For the diagram from Fig. 6 there are two maximal consistent sets of transitions: $T_{C_{max_1}} = \{t_2, t_3\}$, $T_{C_{max_2}} = \{t_2, t_4\}$.

To eliminate potential semantics inconsistencies transitions predicates belonging to different maximal consistent sets must be pairwise non-implicative, i.e. they form *maximal compatibility classes*, where compatibility means non-implicativity relationship.

General condition for non-implicative maximal consistent set of transitions is as follows:

Definition 4: The HCFSM has non-implicative transitions when the following condition is satisfied:

$$\underbrace{\bigwedge_{T_{C_{max}} \subseteq T_z}}_{a} \underbrace{\bigwedge_{t_j, t_k \in T_{C_{max}}}}_{b} \underbrace{\chi_z \not\models t_j \rightarrow t_k \text{ and } \chi_z \not\models t_k \rightarrow t_j}_{c} \tag{12}$$

where t_j and t_k represent predicates of respective transitions, $j \neq k$ and χ_z is a characteristic function of the set of global states.

When broadcasting mechanism is not applied (and some other syntactics features) the context of global states (χ_z) can be left out and calculations can be executed according to the simplified formula:

$$\bigwedge_{T_{C_{max}} \subseteq T_z} \bigwedge_{t_j, t_k \in T_{C_{max}}} t_j \rightarrow t_k \neq \mathbf{1} \text{ and } t_k \rightarrow t_j \neq \mathbf{1} \tag{13}$$

For the given predicates (see equations 9, 10 and 11) their non-implication relationship can be expressed as follows: $T_{C_{max_1}} = \{t_2, t_3\} = \{t_2 : b * \overline{a},\ t_3 : c * \overline{a}\}$

$$t_2 \rightarrow t_3 = b * \overline{a} \rightarrow c * \overline{a} = \overline{b} + a + c * \overline{a} \neq \mathbf{1} \tag{14}$$
$$t_3 \rightarrow t_2 = c * \overline{a} \rightarrow b * \overline{a} = \overline{c} + a + b * \overline{a} \neq \mathbf{1} \tag{15}$$

and $T_{C_{max_2}} = \{t_2, t_4\} = \{t_4 : \overline{a},\ t_2 : b * \overline{a}\}$

$$t_2 \rightarrow t_4 = b * \overline{a} \rightarrow \overline{a} = b + a + \overline{a} = \mathbf{1} \tag{16}$$
$$t_4 \rightarrow t_2 = \overline{a} \rightarrow b * \overline{a} = a + b * \overline{a} \neq \mathbf{1} \tag{17}$$

Because transition t_2 implies transition t_4 ($t_2 \rightarrow t_4$, see eq. 16) the control system with the controller from Fig. 6 can be semantically inconsistent. Changing predicate t_4 for $\overline{a} * \overline{b}$ would remedy this flaw.

Although implicativity formally is correct it breaks modular paradigm.

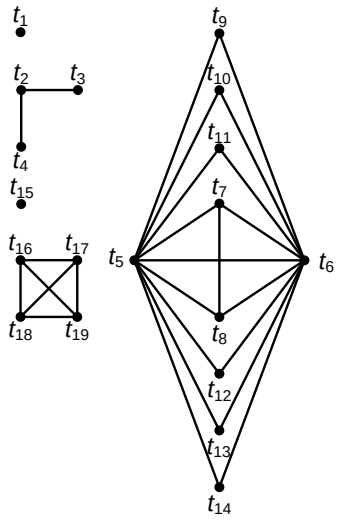


Fig. 7. Orthogonality graph of potentially conflicting transitions.

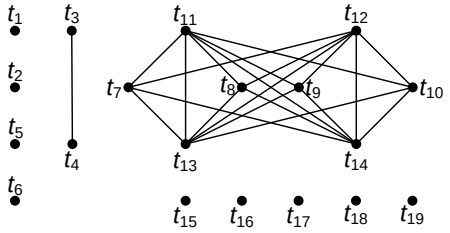


Fig. 8. Non-implicativity graph of concurrent transitions.

V. Predicates Construction Complexities

The main role of formal Hardware Description Languages (*HDL*) is to provide possibilities to exchange information between designer and Computer Aided Design systems (*CAD*). Statechart diagrams also serve as a language for controllers behavior description and at same time as a entry format for CAD systems [12], [17]. The problem of transition predicates construction, as a part of statecharts notation, lies on both computer side and human nature side. From computer point of view the problem comes down to regular computational complexity of the two conditions (definitions 2 and 4). From designer angle the difficulties lie in the fact that propositions of the predicates must be prepared manually. CAD system can only answer the question whether the two conditions are satisfied. Of course, the designer's difficulties are due to computational complexity nature of the problem.

For well designed HCFSM controller definitions 2 and 4 formulate conditions which must be satisfied. Part c of these conditions is a decision problem and presents exponential complexity in terms of number of variables in transition predicates (2^n where n is a number of variables). Checking this part (c) is iteratively executed for transition predicates of different pairs and number of iterations depends on two factors:

a) number of maximal inconsistent set of transitions ($T_{I_{max}}$, definition 2 part a) or number of maximal consistent set of transitions ($T_{C_{max}}$, definition 4 part a),

b) number of transition pairs in maximal inconsistent set of transitions (definition 2 part b) or in maximal consistent set of transitions (definition 4 part b).

The maximal sets of inconsistent and consistent transitions form maximal compatible classes, where compatibility relation is, respectively, orthogonality and non-implicativity. It is well known fact, that compatibility relation can be represented by means of graph called compatibility graph (see Appendix), where vertices correspond to transitions and edges correspond to relations between transitions (i.e. orthogonality and non-implicativity relations, for example see Figs. 7 and 8). Then maximal consistent and inconsistent sets form maximal cliques and the problem of number of maximal consistent/inconsistent sets comes down to the clique problem – the maximal number of cliques possible in a graph with m nodes. In [18] has been proved that any *m*-vertex graph has at most $3^{m/3}$ cliques, so the number of cliques can grow exponentially and hence the number of proper maximal sets.

The number of transition pairs in respective maximal sets (part b of the definitions) can be calculated according to simple formula $\frac{m*(m-1)}{2}$, where m is the number of transitions in the set.

Upper bound of computational complexity of checking the two conditions can be calculated as a product of separate worst case complexities of the tree parts:

$$\underbrace{3^{m/3}}_{a} * \underbrace{\frac{m*(m-1)}{2}}_{b} * \underbrace{2^n}_{c} = O(a^{m+n}) \qquad (18)$$

Part a is a maximal number of maximal cliques possible (maximal consisten/inconsisten set of transitions, m is a number of transitions) in respective compatibility graphs. Part b is a number of edges in a maximum clique. Part c is a complexity of satisfiability problem of *n*-variable Boolean expression. Both sets of transitions (maximally consistent and maximally inconsistent) can easily be constructed through simple search of hierarchy tree (e.g. $T_{I_{max_8}} = \{t_5, t_6, t_7, t_8\}$ and $T_{I_{max_{17}}} = \{t_{16}, t_{17}, t_{18}, t_{19}\}$ in Fig. 3). Figure 9 and Tab. I enumerate these sets.

TABLE I
Family of All Maximal Consistent Transition Sets

$$\begin{aligned}
\mathbf{T}_{C_{max_1}} &= \{t_3, t_4\}\\
\mathbf{T}_{C_{max_2}} &= \{t_{11}, t_{13}, t_7\}\\
\mathbf{T}_{C_{max_3}} &= \{t_{11}, t_{13}, t_8\}\\
\mathbf{T}_{C_{max_4}} &= \{t_{11}, t_{13}, t_9\}\\
\mathbf{T}_{C_{max_5}} &= \{t_{11}, t_{13}, t_{10}\}\\
\mathbf{T}_{C_{max_6}} &= \{t_{11}, t_7, t_{14}\}\\
\mathbf{T}_{C_{max_7}} &= \{t_{11}, t_8, t_{14}\}\\
\mathbf{T}_{C_{max_8}} &= \{t_{11}, t_9, t_{14}\}\\
\mathbf{T}_{C_{max_9}} &= \{t_{11}, t_{10}, t_{14}\}\\
\mathbf{T}_{C_{max_{10}}} &= \{t_{13}, t_7, t_{12}\}\\
\mathbf{T}_{C_{max_{11}}} &= \{t_{13}, t_8, t_{12}\}\\
\mathbf{T}_{C_{max_{12}}} &= \{t_{13}, t_9, t_{12}\}\\
\mathbf{T}_{C_{max_{13}}} &= \{t_{13}, t_{10}, t_{12}\}\\
\mathbf{T}_{C_{max_{14}}} &= \{t_7, t_{12}, t_{14}\}\\
\mathbf{T}_{C_{max_{15}}} &= \{t_8, t_{12}, t_{14}\}\\
\mathbf{T}_{C_{max_{16}}} &= \{t_9, t_{12}, t_{14}\}\\
\mathbf{T}_{C_{max_{17}}} &= \{t_{10}, t_{12}, t_{14}\}
\end{aligned}$$

Computational complexity of the problem is exponential and depends on both the number of transitions (m) and the number

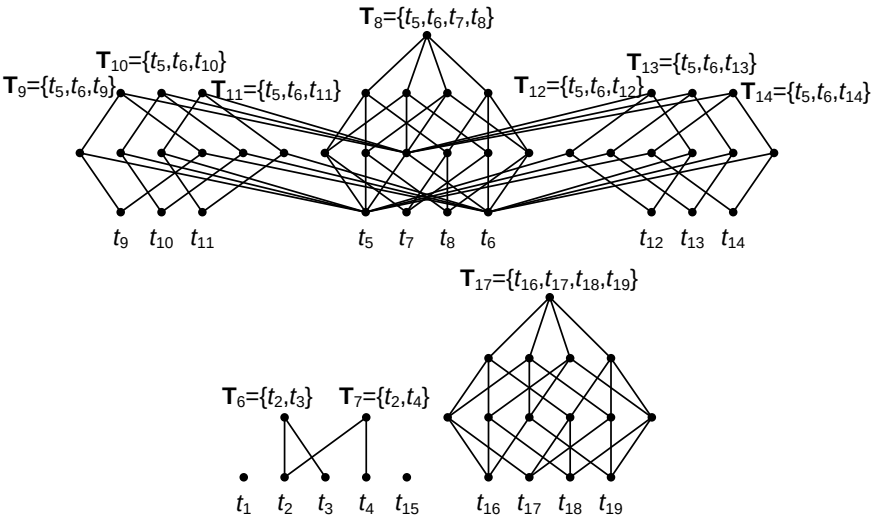


Fig. 9. Hasse diagram of orthogonality graph of potentially conflicting transitions.

of variables in Boolean predicates (n). However, for the designer the problem is hard but feasible, the number of transitions and the cardinality of maximal consitent/inconsistent sets of transitions for typical controller (no bigger than presented in the paper, Fig. 2) makes that it is within human perception.

Good graphical illustrations of the proper sets (maximal consistent/incosistent transition sets) and the scale of the problem is Hasse diagram. Hasse diagram (see Appendix) presents sets and inclusion relation between sets. If we assume that clique is a set of edges representing relation in compatibility graph, then we can say that Hasse diagram presents maximal sets of transitions. Figures 9 and 10 present respective Hasse diagrams for the controller from Fig. 2. Summit vertices mean maximal sets (the sets $T_{I_{max}}$ and $T_{C_{max}}$). The number of maximal inconsistent sets of transition ($T_{I_{max}}$, Fig. 9) is 10 and the number of transitions in maximum set is 4 (Fig. 9). The number of maximal consistent sets of transition ($T_{C_{max}}$, Fig. 10) is 17 and the number of transitions in maximal set is 3 (Fig. 10). Number of transitions for the controller is $m = 19$ and number of Boolean variables in predicates is $n = 10$. All maximal consistent transition sets are showed in Tab. I.

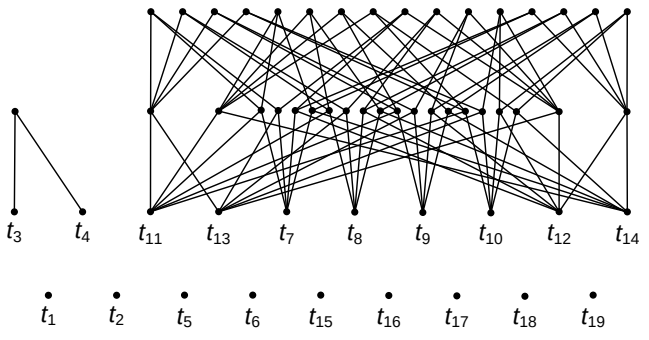


Fig. 10. Hasse diagram of non-implicativity graph of concurrent transitions.

VI. Specification of Well Designed HCFSM

Well designed HCFSM is not only live and safe but also must be conflict-free (transition orthogonality) and concurrent transitions should be independent (transition non-implicativity). Generally, these two conditions can be satisfied through proper construction of transition predicates. However, in case of complex behavior this task can be difficult for the designer. For example, for the diagram from Fig. 2 designer must prepare 18 transition predicates. Next, to make this diagram conflict-free, designer must solve system of 32 Boolean equations made up of this predicates. This number results from definition 2. In order to make that in this diagram transitions are non-implicative designer must solve another system of 49 Boolean equations (see definition 4).

Figure 2 presents statchart diagram of the chemical reactor controller. Transition predicates in this diagram are simplified so as to improve clarity of the drawing; predicates are only equipped with these events which are essential for understanding general idea of the controller operations. Therefore the transitions in diagram are conflicting. One of the possible transitions set of well designed controller is presented in Tab. II. This one particular set of predicates has been achieved after a few tests by „guess and check" method aided with author's CAD system called HiCoS [12], [11].

In general, finding set of correct predicates can be laborious. Using local variable as a synchronizing variable makes that finding correct predicates is much more easier. Improved diagram from Fig. 11 shows that variable of local scope x, y and z (and an *end* states in compound state *Initiating*) make that, in comparison with predicates from Tab. II, respective transitions are not only simpler (e.g. t_8, t_{16}) but process of predicates creation is more algorithmic. Moreover, controller from Fig. 11 has only 41 global states, whereas controller from Fig. 2 (and with transitions from Tab. II) has 161 global states, and both controllers for external observer behave in the same way. Other strategy then presented in this paper is taken by OMG consortium in UML technology [2], where transitions on lower level of hierarchy tree have priority over higher level hierarchy transitions, so vertical conflicts are eliminated and hence extent of computational tasks are decreased.

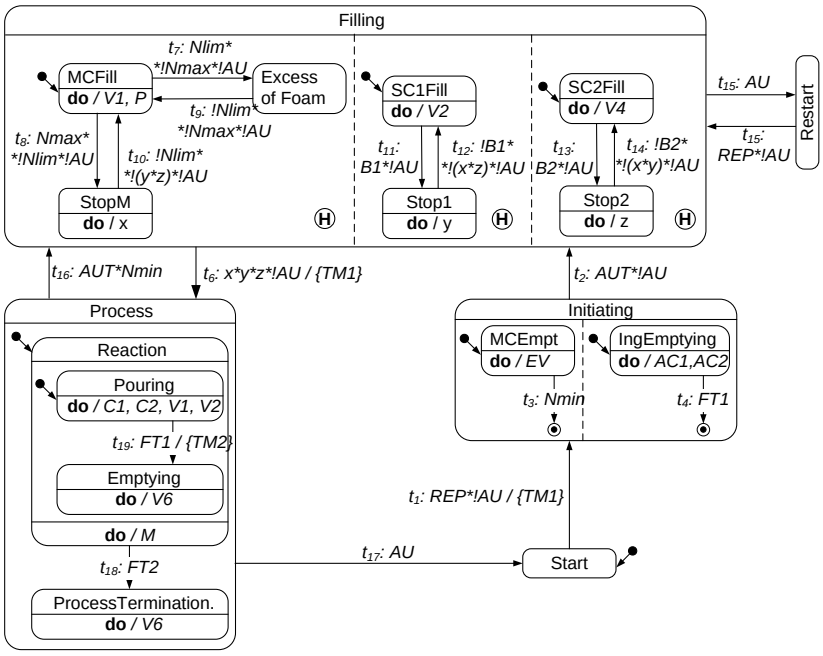


Fig. 11. Diagram improved with local variables synchronization.

TABLE II
THE SET OF WELL DESIGNED TRANSITIONS

$t_1 : REP * \overline{AU}$
$t_2 : AUT * \overline{AU}$
$t_3 : Nmin$
$t_4 : FT1$
$t_5 : AU$
$t_6 : Nmax * \overline{Nlim} * B1 * B2 * \overline{AU}$
$t_7 : Nlim * \overline{Nmax} * \overline{AU}$
$t_8 : Nmax * \overline{Nlim} * \overline{B1} * \overline{B2} * \overline{AU}$
$t_9 : \overline{Nlim} * \overline{Nmax} * \overline{AU}$
$t_{10} : \overline{Nmax} * \overline{AU}$
$t_{11} : B1 * \overline{Nmax} * \overline{AU}$
$t_{12} : \overline{B1} * \overline{AU} * \overline{Nmax}$
$t_{13} : B2 * \overline{Nmax} * \overline{AU}$
$t_{14} : \overline{B2} * \overline{AU} * \overline{Nmax}$
$t_{15} : REP * \overline{AU}$
$t_{16} : AUT * Nmin * \overline{FT1} * \overline{FT2} * \overline{AU}$
$t_{17} : AU$
$t_{18} : FT2 * \overline{FT1} * \overline{AU}$
$t_{19} : FT1 * \overline{AU}$

VII. CONCLUSIONS

Control unit is a vital part of most today's embedded real-time systems. Controllers of complex behavior can efficiently be specified by means of hierarchical and concurrent state oriented notation called statecharts (or state machine) and implemented as a HCFSM. Hierarchy paradigm introduces problems with conflicting transitions, whereas concurrency causes problems with implicative dependencies between transitions what results in inconsistencies in control system. Well designed controller must be both conflict-free and concurrent transitions should be independent. These two properties can be assured through appropriate construction of transition predicates. However, the problem is a clique problem and its size is exponential what makes for a designer that creation of proper predicates can be very hard. Therefore, some syntactic features, like local synchronization variables or end states which make the task easier, can be applied.

APPENDIX A
COMPATIBILITY CLASSES

A given binary relation $\sim$, defined among elements of a set $A = \{a_1, a_2, \ldots, a_n\}$, is said to be an *compatibility relation* [19] iff it is both *reflexive* and *symmetric*, i.e.:

1)

$$\bigwedge_{1 \le i \le n} a_i \sim a_i \quad \text{(reflexivity)},$$

2)

$$\bigwedge_{1 \le i \le n} \bigwedge_{1 \le j \le n} a_i \sim a_j \Rightarrow a_j \sim a_i \quad \text{(symmetry)}.$$

Elements a_i and a_j are said to be *compatible* whenever $a_i \sim a_j$ holds. A *compatibility class* C is a subset of A such that all the members of C are pairwise compatible, i.e.: $a_i, a_j \in C \Rightarrow a_i \sim a_j$. A compatibility class is *maximal* if it is not a proper subset of any compatibility class.

APPENDIX B
COMPATIBILITY GRAPHS

A compatibility relation on a set $A = \{a_1, a_2, \ldots, a_n\}$ can be represented by means of a graph G, called *compatibility graph* [19], where vertices $v_1, v_2, \ldots, v_n$ correspond to elements of A and edges (v_i, v_j) corresponds to compatibility relation whenever $a_i \sim a_j$. A *complete* subgraph of G, called clique, corresponds to compatibility class and maximal clique corresponds to maximal compatibility class.

APPENDIX C
HASE DIAGRAMS

A Hasse diagram [20] is a graphical rendering of a partially ordered set $(S, \leq)$ displayed via the cover relation of the partially ordered set with an implied upward orientation. A point is drawn for each element of the poset, and line segments are drawn between these points according to the following two rules:

1) If $x < y$ in the poset, then the point corresponding to x appears lower in the drawing than the point corresponding to y.
2) The line segment between the points corresponding to any two elements x and y of the poset is included in the drawing iff y covers x and there is no z such that $x < z < y$.

REFERENCES

[1] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design. Modeling, Synthesis and Verification.* Springer, 2009.
[2] *OMG Unified Modeling LanguageTM (OMG UML), Superstructure. Version 2.3*, Object Management Group, OMG, 250 First Avenue, Needham, MA 02494, U.S.A., May 2010. [Online]. Available: http://www.omg.org/spec/UML/2.3/
[3] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems.* Englewood Cliffs, New Jersey: Prentice Hall, 1994.
[4] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, vol. 8, pp. 231–274, 1987.
[5] M. Adamski, "Parallel Controller Implementation using Standard PLD Software." in *FPGAs*, W. Moore and W. Luk, Eds. Abingdon EE&CS Books, Oct. 1991, pp. 296–304.
[6] D. Drusinsky and D. Harel, "Using Statecharts for Hardware Description and Synthesis." *IEEE Transaction on Coputer-Aided Design*, vol. 8, no. 7, pp. 798–807, Jul. 1989.
[7] D. Drusinsky-Yoresh, "A State Assignment Procedure for Single-Block Implementation of State Chart." *IEEE Transaction on Coputer-Aided Design*, vol. 10, no. 12, pp. 1569–1576, Dec. 1991.
[8] S. Ramesh, "Efficient Translation of Statecharts to Hardware Circuits." in *Proceedings of Twelfth International Conference On VLSI Design*, Jan. 1999, pp. 384–389.
[9] *STATEMATE Magnum Code Generation Guide.*, I-Logix Inc., 3 Riverside Drive, Andover, MA 01810 U.S.A., 2001.
[10] K. Buchenrieder, A. Pyttel, and C. Veith, "Mapping statechart models onto an FPGA-based ASIP architecture." in *Proc. EURO-DAC '96*, Sep. 1996, pp. 184–189.
[11] G. Łabiak, "From UML statecharts to FPGA - the HiCoS approach," in *Proceedings of Forum on specification & Design Languages – FDL'03*, Frankfurt am Main, Sep. 2003, pp. 354–363.
[12] HiCos, "HiCoS Homepage," http://www.uz.zgora.pl/~glabiak, 2004. [Online]. Available: http://www.uz.zgora.pl/~glabiak
[13] G. Łabiak, "Transition orthogonality in statechart diagrams and inconsistencies in binary control system," *Przegląd Elektrotechniczny*, no. 9, pp. 130–133, 2010.
[14] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," *ACM Trans. Soft. Eng. Method.*, vol. 5, no. 4, Oct. 1996.
[15] G. Łabiak, "Symbolic States Exploration of UML Statecharts for Hardware Description," in *Design of Embedded Control Systems*, M. A. Adamski, A. Karatkevich, and M. Węgrzyn, Eds. Springer, 2005, pp. 73–83.
[16] A. Maggiolo-Schettini and M. Merro, *Priorities in Statecharts.*, ser. LNCS. Springer–Verlag, 1997, vol. 1192, pp. 404–429.
[17] G. Bazydło, "Behavioural synthesis of reconfigurable controllers based on UML state machine model," *Pomiary, Automatyka, Kontrola*, no. 7, pp. 508–510, 2009, in Polish.
[18] J. W. Moon and L. Moser, "On cliques in graphs," *Israel Journal of Mathematics*, vol. 3, no. 1, pp. 23–28, March 1965.
[19] A. Grasselli, "A note on the derivation of maximal compatibility classes," *Calcolo*, vol. 2, no. 2, pp. 165–176, June 1966.
[20] W. MathWorld, "Wolfram MathWorld," http://mathworld.wolfram.com, 2010. [Online]. Available: http://mathworld.wolfram.com/HasseDiagram.html