Bull. Nov. Comp. Center, Comp.Science, 14 (2001), 7–17

The study of solutions to a parallel FSM equation*

S.A. Buffalov, N.V. Yevtushenko

In this paper, we continue the study of a parallel FSM equation. We establish that the solution set has a lattice structure and consider two restricted solutions to the equation, namely, a supremal and a livelock-free solution.

1. Introduction

When a complex system is designed an important step is to decompose it into a collection of simpler systems which interact in a well-defined manner. The decomposition can be obtained by solving logical equation $A \diamondsuit X \leq C$, if some components are a priori known. Here A describes a behavior of a known part of the system, C is the system specification, $\diamondsuit$ is a composition operator, $\leq$ is a conformance relation, and X is a free variable representing a behavior of an unknown part. The problem is well-known, and a number of publications in process algebra deals with equation solving for labeled transition systems using different conformance relations [1–6].

We limit ourselves with equation solving for Finite State Machines (FSMs), i.e., for labeled transition systems with the set of actions that is divided into two disjoint sets, input and output set. Moreover, every input symbol is followed with an output. As it is shown in [6], the results obtained in process algebra are not directly applicable to the case, since we are interested only in a solution that is a FSM. The problem has been discussed for synchronous [7–9] and parallel [6] composition operators and a reduction relation as the conformance relation. A solvable equation has been shown to have the largest solution that contains any solution as its reduction. Paper [10] generalizes the obtained results for both kinds of the composition operator in the form of FSM languages and expands the results to language equations. In practical situations, some restrictions can be imposed since not each solution is of a practical use. Some restricted solutions and the structure of the corresponding solution set are considered in [10, 11].

In this paper, we continue the study of a parallel FSM equation. Two restricted solutions, namely, supremal and livelock-free are considered. A supremal solution combined with the known part of the system should be defined

*Partially supported by the Russian Foundation for Basic Research under Grant 99-01-00337 and MOPO Grant.

under each input sequence where a behavior of the specification is defined. A livelock-free solution does not allow to have an infinite internal dialog between component FSMs within the designed system.

The structure of the paper is as follows. Section 2 comprises necessary notions of a FSM, a FSM language and a FSM's composition. We show that the solution set has a lattice structure in Section 3. Section 4 deals with supremal and livelock-free solutions.

2. Preliminaries

2.1. Languages and finite automata. An *alphabet* V is a finite set of symbols. We denote V^* the set of all finite sequences over V including the empty sequence ε. A *language* over alphabet V is a subset of V^*. *Prefix* α of word β is denoted $\alpha \preceq \beta$. Given word α in the alphabet V and alphabet W, a *W-restriction* of α, written $\alpha_{\downarrow W}$, is obtained by deleting from α each symbol of the set $V \setminus W$.

In this paper, we limit ourselves with regular languages that are represented by finite automata [12].

A *finite automaton*, often called an automaton throughout this paper, is a quintuple $P = (S, V, \varphi, s_0, F)$, where S is a finite nonempty set of states with initial state s_0 and the subset F of final or accepting states, V is a finite nonempty set of actions, and $\varphi \subseteq S \times (V \cup \varepsilon) \times S$ is a next state relation. If the triple (s, a, s') belongs to φ, then we say there is a transition from the state s to the state s' labeled with action $a \in (V \cup \varepsilon)$.

A sequence $a_1 a_2 \dots a_k \in V^*$ is said to be *accepted* by the automaton P, if there exist sequences $b_1 b_2 \dots b_t \in (V \cup \varepsilon)^*$ and $s_0 s_1 \dots s_t \in S^*$, such that $s_t \in F$, $(s_{i-1}, b_i, s_i) \in \varphi$, $i = 1, \dots, t$, and $a_1 a_2 \dots a_k$ is a V-restriction of $b_1 b_2 \dots b_t$. The set $L(P)$ of all sequences accepted by P is called the *language accepted by automaton* P.

The automaton P is called *deterministic*, if $\varphi \in S \times V \times S$ and for each $s \in S$ and $v \in V$ there exists exactly one state s', such that $(s, v, s') \in \varphi$. Given automaton P, there always exists a deterministic automaton P_d with the same language. We first add a non-accepting state 'DON'T CARE', denoted DNC, to the automaton. If in state $s \in S$ there is no outgoing transition labeled with $v \in V$, then we add the transition (s, v, DNC) to the next state relation, while we add a loop at the state DNC labeled with all $v \in V$. The automaton P_d can be obtained from augmented automaton P by applying subset construction technique [12]. States of P_d are subsets of S. The initial state of P_d is the set comprising s_0 and each state reachable from s_0 via a sequence of transitions labeled with the empty sequence ε. A subset is an accepting state of P_d, if and only if it comprises an accepting state of P. Given subsets $Q, T \subseteq S$ and $v \in V$, the triple (Q, v, T) belongs

to the next state relation of P_d, if and only if $\forall t \in T\ \exists q \in Q\ [(q,v,t) \in \varphi]$ and $\forall q \in Q\ [(q,v,t) \in \varphi \Rightarrow t \in T]$. In most cases, there is no need to apply the full-blown subset construction; it is enough to consider states of P_d that are reachable from its initial state.

We now remind some operations [12], [13] over languages. Regular languages are closed under these operations, and we show how a corresponding deterministic automaton can be derived for the result of each operation. Let $L(P)$ and $L(R)$ be languages of deterministic automata $P = (S, V, \varphi_P, s_0, F_P)$ and $R = (Q, W, \varphi_R, q_0, F_R)$, correspondingly. Without loss of generality, we assume the state sets of P and R be disjoint.

A *union* of languages $L(P)$ and $L(R)$ is a language $L(P) \cup L(R) = \{\alpha \in (V \cup W)^* \mid \alpha \in L(P) \vee \alpha \in L(R)\}$. It is accepted by deterministic automaton $(S \cup Q \cup In, V \cup W, \varphi, In, F_P \cup F_R)$, where $In \notin S \cup Q$. Next state relation φ is obtained by uniting relations φ_P and φ_R and adding the triple (In, v, s) for each transition (s_0, v, s) in P and the triple (In, w, q) for each transition (q_0, w, q) in R.

An *intersection* of languages $L(P)$ and $L(R)$ is a language $L(P) \cap L(R) = \{\alpha \in (V \cap W)^* \mid \alpha \in L(P) \wedge \alpha \in L(R)\}$. It is accepted by deterministic automaton $(S \times Q, V \cap W, \varphi, (s_0, q_0), F_P \cap F_R)$. Given $(s,q), (s',q') \in S \times Q$ and $a \in V \cap W$, the relation φ contains triple $((s,q), a, (s',q'))$, if and only if $(s, a, s') \in \varphi_P$ and $(q, a, q') \in \varphi_R$.

A *complement* of language $L(P)$ is a language $\overline{L(P)} = V^* \setminus L(P)$. A deterministic automaton accepting $\overline{L(P)}$ is obtained from P by interchanging between accepting and non-accepting states.

A *prefix closure* of $L(P)$ is a language $\langle L(P) \rangle = \{\alpha \in V^* \mid \exists \beta \in L(P)\ [\alpha \preceq \beta]\}$. To derive a corresponding deterministic automaton each state of P, from which an accepting state is reachable, is claimed an accepting state. The language $L(P)$ is prefix closed, if $\langle L(P) \rangle = L(P)$.

A *W-restriction* of language $L(P)$ is a language $L(P)_{\downarrow W} = \{\alpha \in W^* \mid \exists \beta \in L(P)\ [\beta_{\downarrow W} = \alpha]\}$. A deterministic automaton accepting language $L(P)_{\downarrow W}$ is derived by replacing each triple $(s, a, s') \in \varphi_P$, where $a \in V \setminus W$, by the triple (s, ε, s'). Then, the obtained automaton is determined.

A *W-expansion* of language $L(P)$ is a language $L(P)_{\uparrow W} = \{\alpha \in (V \cup W)^* \mid \exists \beta \in L(P)\ [\alpha_{\downarrow V} = \beta]\}$. A deterministic automaton accepting the language $L(P)_{\uparrow W}$ is derived from P by including each triple (s, a, s), where $s \in S$ and $a \in W \setminus V$, into the next state relation φ_P.

2.2. Finite state machine (FSM), usually called a machine throughout this paper, is a quintuple $A = \langle S, I, O, T, s_0 \rangle$, where S is a finite nonempty set of states with initial state s_0, I, and O are disjoint input and output alphabets, and $T \subseteq S \times I \times O \times S$ is a transition relation. If $(s, i, o, s') \in T$, then there exists a transition from present state s to next state s' labeled

with input-output pair i/o. If the transition relation is empty, then the FSM is called *trivial*.

The FSM A is *complete*, if for each pair $(s,i) \in S \times I$ there exists a pair $(o,s') \in O \times S$, such that $(s,i,o,s') \in T$. Otherwise, the FSM A is *partial*.

The FSM A is *observable*, if for each triple $(s,i,o) \in S \times I \times O$ there exists at most one state $s' \in S$, such that $(s,i,o,s') \in T$.

The FSM A is *deterministic*, if for each pair $(s,i) \in S \times I$ there exists at most one pair $(o,s') \in O \times S$, such that $(s,i,o,s') \in T$.

The *language of the FSM A at state s* is the set $L_s(A)$ of traces over the alphabet $I \cup O$. The trace $i_0o_0i_1o_1 \ldots i_ko_k$ belongs to $L_s(A)$, if there exists the sequence $s_0s_1 \ldots s_{k+1}$ of states, such that $(s_i, i_i, o_i, s_{i+1}) \in T$, $i = 0, 1, \ldots, k$. By definition, the language $L_s(A)$ contains empty sequence ε. We denote $L(A)$ the language of the FSM A at the initial state, for short. As usual, we say that an input sequence is *defined*, if there is some output sequence produced by the FSM A in response to it, i.e., the input sequence is in the I-restriction of the language $L(A)$. If the FSM A is trivial, then $L(A) = \{\varepsilon\}$. The set of all defined input sequences of the trivial FSM is equal to $\{\varepsilon\}$.

Given defined input sequence αi of partial FSM A, the input sequence is *harmonized*, if for any two traces β and γ from $L(A)$, which I-restrictions are equal to α, $\exists o \in O\ [\beta io \in L(A)]$ implies $\exists o' \in O\ [\gamma io' \in L(A)]$. In other words, we require that independently of what output sequence has been produced in response to α given FSM does have an output response to input sequence αi. By definition, we assume that the empty sequence is harmonized.

The set of all harmonized input sequences of the FSM A can be defined as follows. Given a state s and an input i, we determine a set $s(i)$ of all states that are reachable from the state s under the input i. Let I_s be the set of all inputs common for each state in $s(i)$. Then given a state, pertaining the set $s(i)$, we delete all its transitions induced by an input $i \notin I_s$. The procedure is repeated until no more transition can be deleted.

A FSM language is known to be regular. It is accepted by a deterministic automaton [6] that can be obtained by unfolding each transition $(s,i,o,s') \in T$ into two consecutive transitions labeled with i and o, respectively. However, the converse is not always true, i.e., not each regular language over alphabet $I \cup O$ is a language of an appropriate FSM over input alphabet I and output alphabet O.

Given disjoint alphabets I and O, the language $L \subseteq (IO)^*$ is IO-prefix-closed, if $\alpha io \in L$ implies $\alpha \in L$. Regular language L over the alphabet $I \cup O$ is the language of an appropriate FSM over input alphabet I and output alphabet O, if and only if L is an IO-prefix-closed subset of $(IO)^*$ [10].

Consider an automaton accepting a language of a FSM with input alphabet I and output alphabet O. The set of states of corresponding FSM

is the set of all accepting states of the automaton. The transition relation of the FSM includes quadruple (s, i, o, s'), if and only if there is a sequence of two consecutive transitions from the state s to the state s' labeled with i and o, respectively.

We further denote L^{FSM} the largest IO-prefix-closed subset of $L \cap (IO)^*$.

If L does not include the empty sequence, then L^{FSM} is empty. Otherwise, a deterministic automaton accepting L^{FSM} can be derived in two steps. We first derive an automaton accepting intersection $L \cap (IO)^*$. At the second step, we delete each non-accepting state with an incoming transition labeled with $o \in O$. To minimize an automaton we can iteratively delete each state from which no acceptable state is reachable.

The state s of FSM $A = \langle S, I, O, T_A, s_0 \rangle$ is a *reduction* of the state q of the FSM $B = \langle Q, I, O, T_B, q_0 \rangle$, written $s \leq q$, if $L_s(A) \subseteq L_q(B)$. States s and q are *equivalent*, written $s \cong q$, if $L_s(A) = L_q(B)$. The FSM A is a *reduction* of the FSM B, written $A \leq B$, if the initial state of A is a reduction of the initial state of B. If the initial states of machines A and B are equivalent, then FSMs A and B are *equivalent*, written $A \cong B$. Reduction relation is sometimes called a trace inclusion relation.

The FSM with states that are not pairwise equivalent is called *reduced*. Hereinafter, we consider reduced and observable FSMs, unless otherwise stated.

2.3. FSM composition. Consider a system of two interacting observable FSMs A and B in Figure 1. The FSM A has an input alphabet $I_1 \cup Z$ and output alphabet $O_1 \cup U$. The FSM B has an input alphabet $I_2 \cup U$ and output alphabet $O_2 \cup Z$. For the sake of simplicity, we assume sets I_1, I_2, O_1, O_2, Z, and U be pairwise disjoint. Some of these sets can be empty, provided sets $I = I_1 \cup I_2$ and $O = O_1 \cup O_2$ to be nonempty. The set I is the *set of external inputs*, while the set O is the *set of external outputs*. We refer to the set $Z \cup U$ as to the *set of internal actions*.

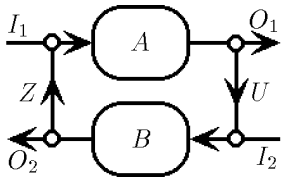

Figure 1. Composition of FSMs A and B

Similar to [6], we further assume that the system has a single message in transit. Moreover, the next external input can be applied only after the system has produced an external output to previous external input. This model of FSM's interaction is often used in protocol engineering and testing and is referred to as parallel composition of FSMs.

For an external observer a single system's execution is described by input-output pair i/o, where $i \in I$ and $o \in O$. The system executes the pair i/o if, for example, there exist internal sequences $z_1 z_2 \ldots z_k \in Z^*$ and $u_1 u_2 \ldots u_k \in U^*$, such that $i u_1 z_1 \ldots u_k z_k o$ is a trace of A, while $u_1 z_1 \ldots u_k z_k$ is that

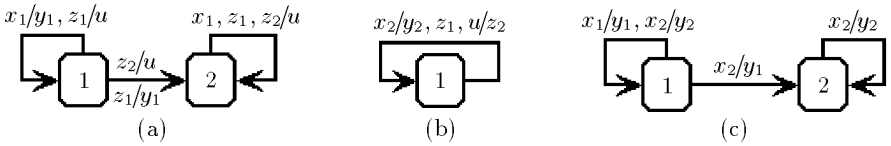

Figure 2. An example of composition (c) of FSMs A (a) and B (b)

of B. In other words, the system has external trace io, if and only if $io \in (L(A)_{\uparrow I_2 \cup O_2} \cap L(B)_{\uparrow I_1 \cap O_1})_{\downarrow I \cap O}$. Thus, we come up to the following definition of the parallel composition [10].

Parallel composition of FSMs A and B is a reduced observable FSM $A \diamondsuit B$ with input alphabet I, output alphabet O, and the language $(L(A)_{\uparrow I_2 \cup O_2} \cap L(B)_{\uparrow I_1 \cap O_1})_{\downarrow I \cap O} \cap (IO)^*$.

Figure 2 shows an example of parallel composition of the FSM A with input alphabet $\{x_1, z_1, z_2\}$ and output alphabet $\{y_1, u\}$ and the FSM B with input alphabet $\{x_2, u\}$ and output alphabet $\{y_2, z_1, z_2\}$. Both have state 1 as initial. Suppose external input x_2 is applied to the composition at the initial state 11. The FSM B can produce either external output y_2 or internal output z_1. In the former case, the system remains at state 11 and can accept the next external input. In the latter case, z_1 is applied to the FSM A and it, for example, produces external output y_1. The system enters state 21 and expects the next external input. Despite of completeness of each component machine, the composed FSM shown in Figure 2c is partial, since there is no transition from state 2 under input x_1. The problem is the component machines fall into infinite internal dialog (i.e., there is a so-called livelock in the composition) $x_1(uz_2)^*$ at state 21 after the system at the initial state has produced y_1 to x_2. Moreover, not each defined input sequence of the composed FSM is harmonized. For example, input sequence x_2x_1 is not harmonized, since there is a trace $x_2y_2x_1y_1$, while there is no trace $x_2y_1x_1y$ for any $y \in \{y_1, y_2\}$. The latter means that a user must be careful applying x_1 after x_2. If the system has produced y_1 to x_2, then the system cannot produce an external output to the next input x_1. We avoid the situation, if only harmonized input sequences are applied to the system. The procedure from Section 2.2 returns the set $\{1, 2\}$ of all states reachable from 1 under x_2. Since at state 2 there is no transitions defined for x_1, we delete at state 1 all transitions induced by x_1. No more transition can be deleted, i.e., the set $(x_2)^*$ is the largest set of harmonized input sequences of the composition.

The language $L_T(A, B) = \langle L(A) \rangle_{\uparrow I_2 \cup O_2} \cap \langle L(B) \rangle_{\uparrow I_1 \cup O_1} \cap \langle (IO)^* \rangle_{\uparrow Z \cup U}$ is called *total language* of the composition $A \diamondsuit B$. By definition, the total language is the set of all traces that may occur within the system of interacting FSMs.

3. Structure of the set of solutions to FSM equation

Let C and A be FSMs with input alphabets $I_1 \cup I_2$ and $I_1 \cup Z$ and output alphabets $O_1 \cup O_2$ and $O_1 \cup U$, respectively. The FSM $A \diamondsuit B$ is called a *decomposition* of C if $A \diamondsuit B \leq C$. The problem of decomposing FSM C can be reduced to solving the FSM equation $A \diamondsuit X \leq C$, where X is a free variable. The FSM B is called a *solution* to the equation, if $A \diamondsuit B \leq C$. As usual, the equation can have no solution or have a set of solutions. In this section, we study the structure of the set of solutions to the equation $A \diamondsuit X \leq C$.

Solution is called the *smallest solution*, if it is a reduction of any other solution. According to definitions, if B is a solution to the equation, then each reduction of B is also a solution to the equation. The trivial FSM is a reduction of each FSM; thus, a solvable equation has the smallest solution, which is trivial in fact.

Theorem 1. *If equation $A \diamondsuit X \leq C$ is solvable, then trivial FSM over alphabets $I_2 \cup U$ and $O_2 \cup Z$ is the smallest solution to the equation.*

The result immediately implies the following

Corollary 1. *The equation $A \diamondsuit X \leq C$ is solvable, if and only if the composition of the FSM A and trivial FSM is a reduction of the FSM C.*

Solution is called the *largest solution*, if each solution is its reduction. A solvable equation is known [6] to have the largest solution, which is a FSM M over alphabets $I_2 \cup U$ and $O_2 \cup Z$ with the language L_M^{FSM} [10], where

$$L_M = \overline{\left[(\overline{L(C)} \cap (IO)^*)_{\uparrow Z \cup U} \cap L(A)_{\uparrow I_2 \cup O_2}\right]_{\downarrow I_2 \cup O_2 \cup Z \cup U}}.$$

An arbitrary FSM is a solution to the equation, if and only if it is a reduction of M, i.e., the largest solution completely describes the set of all solutions to the equation.

We now introduce two binary operations over FSMs. Given FSMs A and B, *intersection* $A \cap B$ is a FSM with the language $L(A) \cap L(B)$, i.e., the intersection is the supremum of two FSMs. The *direct sum* $A \cup B$ is a FSM with the language $L(A) \cup L(B)$, i.e., the direct sum is the infimum of two FSMs. We obtain FSMs $A \cap B$ and $A \cup B$ by converting deterministic automata representing languages $L(A) \cap L(B)$ and $L(A) \cup L(B)$ into corresponding FSMs. The following statement shows that it is always possible.

Proposition 1. *Given FSMs* $A = \langle S, I, O, T_A, s_0 \rangle$ *and* $B = \langle Q, I, O, T_B, q_0 \rangle$, *the languages* $L(A) \cap L(B)$ *and* $L(A) \cup L(B)$ *are FSM languages over alphabets* I *and* O.

In fact, the intersection (union) of two languages that are subsets of $(IO)^*$ also is a subset of $(IO)^*$. Moreover, intersection (union) of two IO-prefix-closed languages also is IO-prefix-closed. Thus, $L(A) \cap L(B)$ and $L(A) \cup L(B)$ are languages of appropriate FSMs.

By definition, $A \cap B \leq B$, i.e., given solutions B and D to the equation $A \diamondsuit X \leq C$, FSM $B \cap D$ also is a solution. On the other hand, for each trace α of the FSM B or D it holds that $[L(A)_{\uparrow I_2 \cup O_2} \cap \{a\}_{\uparrow I_1 \cup O_1}]_{\downarrow I \cup O} \cap (IO)^*$ is a subset of $L(C)$. Thus, the FSM $B \cup D$ also is a solution to the equation.

Proposition 2. *Given solutions* B *and* D *to the equation* $A \diamondsuit X \leq C$, *intersection* $B \cap D$ *and direct sum* $B \cup D$ *also are solutions to the equation.*

Thus, the set of solutions has a lattice structure.

Theorem 2. *The set of solutions to solvable equation* $A \diamondsuit X \leq C$ *has a lattice structure.*

4. Supremal and livelock-free solutions

In practical situations some restrictions can be imposed, since not each solution is of a practical use. We first notice that usually applying next input a user does not care which output has been produced to the former input sequence, i.e., we further assume each defined input sequence of specification C is harmonized. Secondly, each real implementation is a completely specified FSM, since it produces some output response to any input sequence. If we do not care of an implementation for an undefined transition, then partial FSMs A and C become complete after adding a designating state 'DON'T CARE' (DNC) with loops for each input-output pair. For every input i, such that at state s no transitions is defined under i, we add a transition (s, i, o, DNC) for every output o. After such 'don't care' interpretation of undefined transitions a new equation $A^a \diamondsuit X \leq C^a$ can be solved for augmented FSMs A^a and C^a. Any complete solution B to the equation $A^a \diamondsuit X \leq C^a$ can be used together with any complete implementation Imp_A of FSM A; it is guaranteed that the set of output responses of composition $\text{Imp}_A \diamondsuit B$ to any defined input sequence of the FSM C is contained in that of the FSM C.

Figure 3 illustrates the procedure for equation solving. The set of all defined input sequences of the specification C (Figure 3a) is $(x_1, x_2)^*$. The context A (Figure 3b) is partial. In order to have a solution for any complete

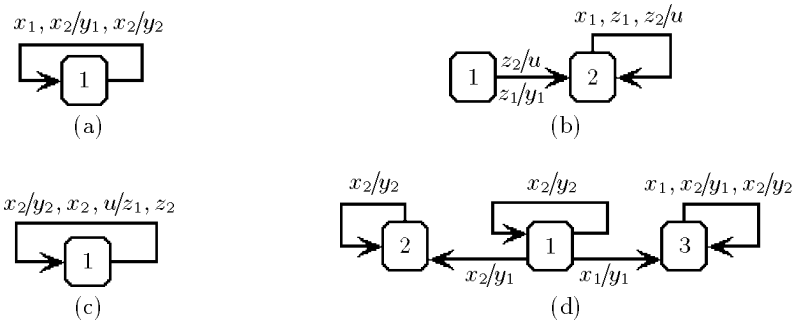

Figure 3. An example of equation solving

implementation of the FSM A we augment it with a designated DNC state. Augmented FSM A^a is obtained by adding at state 1 two transitions to the DNC state labeled with pairs x_1/y_1 and x_1/u and loops at the DNC state labeled with all input-output pairs. The largest solution M (Figure 3c) to the equation $A^a \diamondsuit X \leq C^a$ is a complete FSM. By direct inspection, one can assure that the FSM $A^a \diamondsuit M$ (Figure 3d) is a reduction of C. Thus, any complete reduction of the FSM M combined with any complete implementation of the context is a decomposition of the FSM C.

However, the largest solution can be partial. In this case, we determine the largest complete submachine M_C of the largest solution M [6]. If submachine M_C does not exist, then there is no complete solution to the equation. Otherwise, a complete FSM is a solution to the equation, if and only if it is a reduction of M_C. Given complete solution B to the equation $A \diamondsuit X \leq C$, the composition of two complete FSMs A and B can return a partial FSM, as it was demonstrated by an example (see Figure 2). However, in practical situations we are going to apply to the composition only input sequences, where a behavior of the specification C is defined. In other words, we need a solution B, such that the set of defined harmonized input sequences of composition $A \diamondsuit B$ contains that of the FSM C. We call the solution B with the above property a supremal solution.

Given solvable equation $A \diamondsuit X \leq C$ and the set $\Im$ of input sequences of the FSM C, the solution B to the equation is $\Im$-*available*, if the set of all harmonized input sequences of the FSM $A \diamondsuit B$ contains $\Im$. We call B a *supremal solution*, if B is $\Im_C$-available, where $\Im_C$ is the set of all defined input sequences of the FSM C.

If the largest solution is supremal, then it is the largest supremal solution. However, not each reduction of the largest solution inherits the property. When the largest solution is not supremal, the equation still can have a supremal solution. In this case, the largest solution could be trimmed until we obtain a supremal solution or establish an absence of a supremal solution. However, it is an open question whether the largest supremal solution exists.

As we demonstrated above, component machines can fall into infinite internal dialog when an appropriate external input sequence is applied to the system. Constructing a system, the designer usually is required to avoid livelocks, at least when a defined input sequence of the specification C is applied. Thus, we come up to the notion of a so-called $\Im$-livelock-free solution.

Given solution B to the equation $A \diamondsuit X \leq C$ and the set $\Im$ of input sequences of the FSM C, the solution B is *$\Im$-livelock-free*, if for each input sequence $\alpha \in \Im$, the total language $L_T(A, B)$ of the composition of A and B does not contain an infinite set of the sequences with $(I_1 \cup I_2)$-restriction α. A solution is *livelock-free*, if it is $\Im_C$-livelock-free, where $\Im_C$ is the set of all defined input sequences of the FSM C.

If the largest solution is livelock-free, then it is the largest livelock-free solution to the equation. Moreover, each its reduction also is a livelock-free solution. However, similar to a supremal solution, a livelock-free solution can exist, when the largest solution is not livelock-free. Given the example in Figure 3, the largest solution is neither supremal, nor livelock-free.

5. Conclusion

In this paper, we continue to study properties of solutions to a parallel FSM equation. We have shown that the set of all solutions to the equation has a lattice structure. In the case, when the specification is partial, we have considered two restricted solutions, a supremal and a livelock-free solution. It should be noticed that a composed system is a safe implementation of the specification, if and only if we use a supremal and livelock-free solution. By this reason, both solutions are of a practical interest and need additional research. Just now the question whether there exist the largest supremal and/or livelock-free solution remains open.

References

[1] Larsen Kim G., Xinxin Liu. Compositionality through an operational semantics of contests // J. Logic Computation. – 1991. – Vol. 1, № 6. – P. 761–795.

[2] Wonham W.M., Ramadge P.J. On the supremal controllable sublanguage of a given language // SIAM J. Control. Optim. – 1987. – Vol. 25, № 3. – P. 637–659.

[3] Qin H., Lewis P. Factorization of finite state machines under strong and observational equivalencies // Formal Aspects of Computing. – 1991. – P. 284–307.

[4] Ramadge P.J., Wonham W.M. Supervisory control of discrete event processes // Feedback Control of Linear and Nonlinear Systems / Eds. D. Hinrich-

sen, A. Isidory. Lecture Notes on Control and Information Sciences. – Berlin: Springer-Verlag, 1982. – № 39. – P. 202–214.

[5] Barrett G., Lafortune S. Bisimulation, the supervisory control problem and strong model matching for finite state machines // Discrete Event Dynamic Systems: Theory and Application. – 1998. – Vol. 8, № 4. – P. 377–429.

[6] Petrenko A., Yevtushenko N. Solving asynchronous equations // IFIP TC6 WG6.1 Joint International Conference, Paris, France, 3–6 November, 1998. – P. 231–247.

[7] Kam T., Villa T., Brayton R., Sangiovanni-Vincentelli A. Synthesis of Finite State Machines: Functional Optimization. – Boston: Kluwer Academic Publishers, 1997.

[8] DiBenedetto M.D., Saldanha A., Sangiovanni-Vincentelli A. Model matching for finite state machines // Proc. 33rd Conf. Decision and Control (Lake Buena Vista, FL, USA), December, 1994. – P. 3117–3124.

[9] Khatri S.P., Narayan A., Krishnan S.C., McMillan K.L., Vincentelli A., Brayton R.K. An Engineering Change Methodology Using Simulation Relations. – Berkeley, 1999. – (Technical Report / University of California; March 19. ERL-95-50).

[10] Yevtushenko N., Villa T., Brayton R.K., Petrenko A., Sangiovanni-Vincentelli A. Logic synthesis by equation solving // Proc. of the XVI Intern. Workshop on Logic Synthesis, USA, 2000. – P. 11–14.

[11] Yevtushenko N., Buffalov S. Solving a parallel FSM equation // Discretnaya Matematika, to appear (in Russian).

[12] Hopcroft J.E., Ullman J.D. Introduction to Automata Theory, Languages, and Computation. – Addison-Wesley, 1979.

[13] Rozenberg G., Salomaa A. Handbook of Formal Languages. – Berlin; New York: Springer-Verlag, 1997.