SOLUTION OF SYNCHRONOUS LANGUAGE EQUATIONS FOR LOGIC SYNTHESIS

N.Yevtushenko[*], T.Villa[**], R.Brayton[***], A.Petrenko[****], A.Sangiovanni-Vincentelli[***]

[*]*Tomsk State University, Tomsk, Russia*
[**]*Parades, Roma, Italy*
[***]*California University, Berkeley, USA*
[****]*CRIM Montreal, Canada*

Consider the problem to design a component that combined with a known part of a system, called the context, conforms to a given overall specification. We cast the problem as solving an abstract equation over languages. In this paper, we address only synchronous language equation. We study the most general solutions to the language equation, defining the language operators needed to express them. Successfully we specialize such language equation associated to important classes of automata used for modeling systems; e.g., regular languages and FSM equations. In particular, we show how to compute the largest FSM language that is a solution to the language equation, the largest complete solution, and the largest solution whose composition with the context A yields the language of a complete FSM.

Introduction

An important step in the design of complex systems is the decomposition of the system into a number of separate components, which interact in some well-defined way. In this context, a typical question is how to design a component that combined with a known part of the system, called the context, conforms or satisfies or matches a given overall specification. This question arises in several applications ranging from logic synthesis to the design of discrete controllers; the problem has been studied for various network topologies. There are three key issues to consider:

- how to model the system and its components and the specification;
- how to model the composition of the components, and
- how to model the notion of a system conforming or satisfying or matching another system.

In [1, 2] we proposed a general frame based on defining equations over languages associated to the components of a given system. We introduced two composition operators for abstract languages: synchronous composition, •, and parallel composition, ◊, and we checked conformity by language containment. Then we studied the most general solutions to the language equations, defining the language operators needed to express them. We also specialized the theory of equations over languages to the languages associated to finite automata (FA) and finite state machines (FSM). In particular, we studied the solutions to the equations defined over FSMs of the type $M_A \bullet M_X \subseteq M_C$ and $M_A \lozenge M_X \subseteq M_C$, where M_A models the context, M_C models the specification and M_X is unknown.

In this paper, we pursue further this line of investigation and present only the theory of synchronous equations, i.e., those of the type $M_A \bullet M_X \subseteq M_C$. We refer to [3] for a report on FSM equations of the type $M_A \lozenge M_X \subseteq M_C$.

1. Equations over languages

An alphabet is a finite set of symbols. The set of all finite strings over a fixed alphabet X is denoted by X^*. X^* includes the empty string ε. A subset $L \subseteq X^*$ is called a *language* over alphabet X. In usual way, standard operations on languages are defined: the *union*, the *intersection*, the *complement* and the *difference*. The language L is *prefix-closed* if each prefix of each word is in the language L. A *substitution* [4] f is a mapping of an alphabet X onto subsets of Y^* for some alphabet Y. The substitution f is extended to strings by setting $f(\varepsilon) = \varepsilon$ and $f(\alpha x) = f(\alpha)f(x)$.

Given a language L over alphabet $X \times V$, consider the substitution h defined as $h(x,v) = x$ for all $x \in X$ and $v \in V$. Then the language $L_{\downarrow X} = \{h(\alpha)\colon \alpha \in L\}$ is the *projection* of language L onto alphabet X or *X-projection* of language L. Given a language L over alphabet X and alphabet V, let f be the substitution such that $f(x) = \{(x,v)\colon v \in V\}$ for all $x \in X$. Then the language $L_{\uparrow V} = \{f(\alpha)\colon \alpha \in L\}$ is the *lifting* of language L over alphabet V or *V-lifting* of language L. The following straightforward fact holds between the projection and lifting operators.

Proposition 2.1. Given alphabets X and V, a language L over alphabet X and a string $\alpha \in (X \times V)^*$, $\alpha_{\downarrow X} \in L$ iff $\alpha \in L_{\uparrow V}$.

We introduce some classes of languages used later in the paper.

Definition 2.1. A language L over alphabet $X = I \times O$ is *I-progressive* if $\forall \alpha \in X^* \; \forall i \in I \exists o \in O [\alpha \in L \rightarrow \alpha(i,o) \in L]$.

Definition 2.2. A language L over alphabet $X = I \times O$ is *I-defined* if $L_{\downarrow I} = I^*$.

If a language over $X = I \times O$ is I-progressive then it is also I-defined, but the converse does not hold.

Definition 2.3. A language L over alphabet $X = I \times O$ is *deadlock-free* if $\forall \alpha \in X^* \; \exists \beta \in X^* \setminus \{\varepsilon\} \; [\alpha \in L \rightarrow \alpha\beta \in L]$.

If a language over $X = I \times O$ is I-progressive then it is also deadlock-free, but the converse does not hold.

Definition 2.4. A language L over alphabet $X = I \times O$ is *Moore if* $\forall \alpha \in L \forall (i,o) \in X \; \forall (i',o') \in X \; [\alpha(i',o') \in L \rightarrow \alpha(i',o) \in L]$.

Consider two systems A and B with associated languages $L(A)$ and $L(B)$. The systems communicate with each other by a channel U and with the environment by channels I and O. We introduce a composition that describes the external behavior of the composition of $L(A)$ and $L(B)$.

Definition 2.5. Given alphabets I, U, O, language L_1 over $I \times U$ and L_2 over $O \times U$, the *synchronous composition* of L_1 and L_2 is the language[1] $[(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}]_{\downarrow I \times O}$, written $L_1 \bullet L_2$, defined over $I \times O$.

In the sequel it will be useful to consider the notions of I-progressive and deadlock-free languages extended to the composition of two languages.

Definition 2.6. Given a language A over alphabet $I \times U$, a language B over alphabet $U \times O$ is *A-compositionally I-progressive* if the language $L = A_{\uparrow O} \cap B_{\uparrow I}$ over $X = I \times U \times O$ is I-progressive, i.e., $\forall \alpha \in X^* \; \forall i \in I \exists (u,o) \in U \times O [\alpha \in L \rightarrow \alpha(i,u,o) \in L]$.

Definition 2.7. Given a language A over alphabet $I \times U$, a language B over alphabet $U \times O$ is *A-compositionally deadlock-free* if the language $L = A_{\uparrow O} \cap B_{\uparrow I}$ over $X = I \times U \times O$ is deadlock-free, i.e., $\forall \alpha \in X^* \; \exists \beta \in X^* \setminus \{\varepsilon\} \; [\alpha \in L \rightarrow \alpha\beta \in L]$.

When clear from the context, instead of A-compositionally we will write more simply *compositionally*.

Given alphabets I, U and O, a language A over alphabet $I \times U$ and a language C over alphabet $I \times O$, let us consider the language equation $A \bullet X \subseteq C$.

Definition 2.8. Given alphabets I, U and O, a language A over alphabet $I \times U$ and a language C over alphabet $I \times O$, language B over alphabet $U \times O$ is called a *solution* to the equation $A \bullet X \subseteq C$ iff $B \neq \emptyset$ and $A \bullet B \subseteq C$. The *largest solution* is the solution that contains any other solution.

Theorem 2.1. *The largest solution to the equation $A \bullet X \subseteq C$ is the language $S = \overline{A \bullet \overline{C}}$ if $S \neq \emptyset$.*

Proof. Consider a sequence $\alpha \in (U \times O)^*$, then α is in the largest solution to $A \bullet X \subseteq C$ iff $A \bullet \{\alpha\} \subseteq C$ and the following chain of equivalence follows:

$$A \bullet \{\alpha\} \subseteq C \Leftrightarrow (A_{\uparrow O} \cap \{\alpha\}_{\uparrow I})_{\downarrow I \times O} \cap \overline{C} = \emptyset \Leftrightarrow A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U} = \emptyset \Leftrightarrow$$

$$\alpha \notin (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow O \times U} \Leftrightarrow \alpha \in \overline{A \bullet \overline{C}}.$$

Therefore, the largest solution to the equation $A \bullet X \subseteq C$ is given by *language* $S = \overline{A \bullet \overline{C}}$ if $S \neq \emptyset$.

Corollary 2.1. A language $B \neq \emptyset$ over alphabet $U \times O$ is a solution to $A \bullet X \subseteq C$ iff $B \subseteq \overline{A \bullet \overline{C}}$. If $\overline{A \bullet \overline{C}} = \emptyset$ then the language equation has no solution.

Let S be the largest solution to the equation $A \bullet X \subseteq C$. It is of interest to investigate subsets of S that satisfy some further properties, e.g., of being prefix-closed, progressive, etc.

---

[1] 1. Use the same order $I \times U \times O$ in $(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}$. 2. The projection can be taken on the set that is different from $I \times O$.

If S is prefix-closed then S is the largest prefix-closed solution to the equation. However, not each non-empty subset of S inherits the feature of being prefix-closed. If S is not prefix-closed then denote by $Pref(S)$ the largest prefix-closed subset of S. The set $Pref(S)$ is obtained from S by deleting each string that has a prefix not in S.

Proposition 2.2. If $Pref(S)\neq\emptyset$ then $Pref(S)$ is the largest prefix-closed solution to the equation $A\bullet X\subseteq C$. If $Pref(S)=\emptyset$ then the equation has no prefix-closed solution. If the language S does not include the empty string then the equation has no prefix-closed solution.

If S is U-progressive (S is a language over alphabet $U \times O$) then S is the largest U-progressive solution to the equation. However, not each non-empty subset of S inherits the feature of being U-progressive. If S is not U-progressive then denote by $Prog(S)$ the largest U-progressive subset of S. The set $Prog(S)$ is obtained from S by deleting each string α such that, for some $u\in U$, there is no $o\in O$ for which it holds $\alpha(u,o)\in S$.

Proposition 2.3. If $Prog(S) \neq \emptyset$ then the language $Prog(S)$ is the largest U-progressive solution to the equation $A\bullet X\subseteq C$. If $Prog(S) = \emptyset$ then the equation has no U-progressive solution.

If S is deadlock-free then S is the largest deadlock-free solution to the equation. However, not each non-empty subset of S inherits the feature of being deadlock-free. If S is not deadlock-free then denote by $Dlockf(S)$ the largest deadlock-free subset of S. The set $Dlockf(S)$ is obtained from S by deleting each string α such that there is no $\beta\in(U \times O)^{\bullet}\setminus\{\varepsilon\}$ for which it holds $\alpha\beta\in S$.

Proposition 2.4. If $Dlockf(S) \neq \emptyset$ then the language $Dlockf(S)$ is the largest deadlock-free solution to the equation $A\bullet X\subseteq C$. If $Dlockf(S) = \emptyset$ then the equation has no deadlock-free solution.

If S is compositionally deadlock-free then S is the largest compositionally deadlock-free solution to the equation. However, not each non-empty subset of S inherits the feature of being compositionally deadlock-free. If S is not compositionally deadlock-free then denote by $cDlockf(S)$ the largest compositionally deadlock-free subset of S. The set $cDlockf(S)$ is obtained from S by deleting each string α such that there is no $\beta\in(I \times U \times O)^{\bullet}\setminus\{\varepsilon\}$ for which it holds $\alpha\beta\in A_{\uparrow O}\cap S_{\uparrow I}$.

Proposition 2.5. If $cDlockf(S) \neq \emptyset$ then the language $cDlockf(S)$ is the largest compositionally deadlock-free solution to the equation $A\bullet X\subseteq C$. If $cDlockf(S) = \emptyset$ then the equation has no compositionally deadlock-free solution.

Language equations can be effectively solved when they are defined over languages that can be manipulated algorithmically. Usually such languages are presented through their corresponding automata. In the following sections we are going to study equations over various classes of automata, like FAs and FSMs, specializing the theory of equations to their associated languages. Notice that a key issue to investigate is the closure of the solution set with respect to a certain type of language, e.g., when dealing with FSM language equation we require the solutions are FSM languages too. This cannot be taken for granted, because the general solution of abstract language equation is expressed through the operators of complementation and composition that do not preserve certain classes of languages.

2. Equation over Finite Automata

Definition 3.1. *A non-deterministic finite automaton (NDFA* or simply *FA)* is defined as a 5-tuple $F_A = (S, \Sigma, \Delta, r, F)$. S represents the finite state space, Σ represents the alphabet, and $\Delta\subseteq\Sigma\times S\times S$ is the next state relation. The initial or reset state is $r\in S$ and $F\subseteq S$ is the set of final or accepting states.[2] The next state relation can be extended to have as argument strings in $\Sigma^{\bullet}$ in usual way. A string x is said to be *accepted* by the FA F_A if there exists state $s'\in F$ and a sequence of transitions corresponding to x that takes the FA from the initial state to s'. The language *accepted* by F_A, denoted $L(F_A)$, is the set of all strings accepted by F_A. If for each present state p and symbol i there exists at least one next state n such that $(i, p, n)\in\Delta$ then the FA is said to be *complete*. A FA is a deterministic finite automaton (DFA) if for each present state p and symbol i there is exactly one next state n such that $(i, p, n)\in\Delta$. In the DFA the relation Δ can be replaced by the next state function δ.

The languages accepted by finite automata are the *regular* languages. Regular languages are closed under union, concatenation, complementation and intersection. Regular languages are closed also under projection and lifting, because they are closed under substitution [4].

Two well-known results are non-deterministic finite automata are equivalent to deterministic ones and that regular languages and only them are accepted by finite automata. By applying the algorithm of subset construction one converts a NDFA into an equivalent DFA that also is complete. To make complete a FA it suffices to add a new non-accepting state whose incoming transitions are (i, s, s_d) for all i, s for which there was no transition in the original automaton. By a closure construction [4], an NDFA with ε–moves can be converted to an FA without ε–moves. To show regular languages are closed under the above operators a constructive procedure is proposed that yields the finite automaton of the result, given finite automata of the arguments. For the most common operators see [4]. Here

[2] A variant of NDFA allows the introduction of ε-moves, meaning that $\Delta\subseteq(\Sigma\cup\varepsilon)\times S\times S$.

we only sketch the procedure for lifting. Given FA F_A that accepts language L over X, FA F'_A that accepts language $L_{\uparrow V}$ over $X \times V$ is obtained from by replacing each edge (x, s, s') by the edges $((x,v), s, s')$, $v \in V$.

Given that all the operators used to express the solution of regular language equations have constructive counterparts on automata, we conclude that there is an effective way to solve equations over regular languages. Given a regular language equation $A \bullet X \subseteq C$, where A is a regular language over alphabet $I \times U$, C is over $I \times O$, and the unknown language X is over $U \times O$, we use the corresponding operators over automata to build the automaton with the language $\overline{A \bullet \overline{C}}$[3].

3. Equations over Finite State machines

Definition 4.1. A *non-deterministic FSM* (*NDFSM*) or simply an *FSM* or a *machine*, is defined as a 5-tuple $M = (S, I, O, T, r)$. S represents the finite state space, I represents the finite input space, O represents the finite output space and $T \subseteq I \times S \times S \times O$ is the transition relation. *State* $r \in S$ represents the initial or reset state. If at least one transition is specified for each present state and each input, the FSM is said to be *complete*; otherwise, the FSM is *partial*. An FSM is said to be *trivial* if $T = \emptyset$. The transition relation of a NDFSM can be extended in the usual way to a relation on $I^* \times S \times S \times O^*$. An FSM is *deterministic* (*DFSM*) if for each pair (i, p) there exists at most one next state n and output o such that $(i, p, n, o) \in T$. In deterministic FSM the transition relation usually is replaced by two functions: *next state function* δ and *output function* λ. A complete FSM is said to be of *Moore* type if $(i, p, n, o) \in T$ implies that there is n' such that $(i', p, n', o) \in T$[4]. An FSM is *observable* [5] if for each triple (i, p, o) there exists at most one next state such that $(i, p, n, o) \in T$. In this paper, FSMs are assumed to be observable, unless otherwise stated. Notice that it is always possible to convert a general NDFSM into an observable FSM by subset construction.

We now introduce the notion of language associated to an FSM. This is achieved by looking to the automaton underlying a given FSM. For our purposes, we define an associated language over alphabet $I \times O$. In this case, the automaton coincides with the original FSM where all states are made accepting and the edges carry a label of the type (i, o).

Definition 4.2. Given an FSM $M = (S, I, O, T, r)$, consider the finite automaton $F(M) = (S, I \times O, \Delta, r, S)$, where $((i,o)\ s, s') \in \Delta$ iff $(i, s, s', o) \in T$. The language accepted by $F(M)$ is denoted $L_r(M)$, and by definition is the *language* of M at state r. Similarly, $L_s(M)$ denotes the language accepted by $F(M)$ when started at state s, and by definition is the language of M at state s. The empty string $\varepsilon \in L_s(M)$ because each state is accepting. An FSM is trivial iff $L_r(M) = \varepsilon$.

Definition 4.3. A language L over $I \times O$ is an FSM language if there is an FSM (S, I, O, T, r) such that the associated automaton accepts L.

When convenient, we will say FSM M_A has property X if its associated FSM language has property X.

Definition 4.4. State t of FSM M_B is said to be a *reduction* of state s of FSM M_A, written $t \leq s$, if $L_t(M_B) \subseteq L_s(M_A)$. States t and s are said to be *equivalent* if $t \leq s$ *and* $s \leq t$, i.e., when $L_t(M_B) = L_s(M_A)$. States t and s that are not equivalent are called *distinguishable*. An FSM whose states are distinguishable is called a *reduced* FSM. Similarly, M_B is a *reduction* of M_A, $M_B \leq M_A$, if the initial state of M_B is a reduction of the initial state of M_A. When $M_B \leq M_A$ and $M_A \leq M_B$ then M_B and M_A are *equivalent machines*, i.e. $M_B \cong M_A$. Equivalent machines have equal languages.

For complete DFSMs the reduction and equivalence relations coincide. Given an FSM language, there is a family of equivalent FSMs associated with it. In this paper, FSMs are assumed to be reduced, unless otherwise stated. An FSM language is regular, whereas the converse is not true.

Theorem 4.1. A regular language over alphabet $I \times O$ is the language of a complete FSM over input alphabet I and output alphabet O iff L is prefix-closed and I-progressive. A regular language that is prefix-closed but not I-progressive is the language of a partial FSM.

Given a regular language over alphabet $I \times O$, an algorithm follows to build L^{FSM}, the largest subset of L that is the language of an FSM over input alphabet I and output alphabet O. We first build a deterministic automaton accepting the language L. The automaton accepting the largest prefix-closed subset of L is obtained by deleting all non-accepting states with their incoming edges. If the initial state is deleted then $L^{FSM} = \emptyset$. Otherwise, the resulting automaton accepts an FSM language L^{FSM}. To obtain the largest subset of L^{FSM} that is the language of a complete FSM we iteratively delete from the automaton all states that have an undefined transition at least for one input. If the initial state is deleted then $Prog(L^{FSM}) = \emptyset$. Otherwise, the automaton accepts the largest I-progressive subset $Prog(L^{FSM})$ of L^{FSM}. An FSM whose language is L^{FSM} or $Prog(L^{FSM})$ can be trivially deduced from the automaton by interpreting each label (i,o) as an input/output pair i/o.

Theorem 4.2. Given a regular language over alphabet $I \times O$, let M be an (complete) FSM over input alphabet I and output alphabet O. The language $L(M)$ of M is contained in L if and only if $L(M) \subseteq L^{FSM}$ ($L(M) \subseteq Prog(L^{FSM})$).

---

[3] Notice the procedure can be applied for any regular languages, not only for prefix-closed languages as in restricted versions reported in the literature.

[4] Notice this definition is different from that introduced in [5].

Finally, we discuss how the largest Moore submachine *Moore(M)* of an FSM M can be derived. Given state s of machine M, the set $K_s = \{o \in O | : \forall o \in O\ \exists s' \in S$ s.t. $(i, s, s', o) \in T_M\}$ is defined. We then iteratively delete from M each state s such that $K_s = \emptyset$ with its incoming edges. If $K_s \neq \emptyset$ delete from T_M each transition (i, s, s', o) such that $o \notin K_s$. If the initial state has been deleted then *Moore(M)* does not exist; otherwise, we denote *Moore(M)* the obtained FSM.

Theorem 4.3. *Any Moore FSM M' that is a reduction of M is a reduction of Moore(M).*

Different types of composition between pairs of FSMs can be defined, according to the protocol by which signals are exchanged. For a given composition operator and pair of FSMs we must establish whether the composition of these two given FSMs is defined, meaning that it yields a set of behaviors that can be described by another FSM. So in general, the composition of FSMs is a partially specified function from pairs of FSMs to an FSM. In this paper, we define the composition of FSMs by means of the synchronous composition operator over languages introduced in Section 2. So the FSM yielded by the composition of FSMs M_A and M_B is the one whose language is obtained by the composition of the FSM languages associated to M_A and M_B. The synchronous composition operator models the synchronous connection of sequential circuits.

Consider the pairs of FSMs: FSM M_A with input alphabet $I_1 \times V$, output alphabet $U \times O_1$ and transition relation T_A and FSM M_A with input alphabet $I_2 \times U$, output alphabet $V \times O_2$ and transition relation T_B. We define a synchronous composition operator $\bullet$ that associates to a pair of FSMs another FSM such that the external input alphabets $I = I_1 \times I_2$ while the external output alphabet is $O = O_1 \times O_2$. We remind that, by definition of synchronous composition of languages, a sequence $\alpha \in (I_1 \times I_2 \times O_1 \times O_2)^*$ is in the language of the synchronous composition of $L(M_A)$ and $L(M_B)$ iff α is in the projection onto $I_1 \times I_2 \times O_1 \times O_2$ of the intersection of the liftings, respectively, of $L(M_A)$ over $I_2 \times O_2$ and of $L(M_B)$ over $I_1 \times O_1$[5], i.e. $\alpha \in [L(M_A)\uparrow_{I2\times O2} \cap L(M_B)\uparrow_{I1\times O1}]\downarrow_{I\times O}$. Notice that the liftings $L(M_A)\uparrow_{I2\times O2}$ and $L(M_B)\uparrow_{I1\times O1}$ are needed to have the languages of M_A and M_B defined on the same alphabet $I_1 \times I_2 \times U \times V \times O_1 \times O_2$; e.g., $L(M_B)$ is defined over $I_2 \times U \times V \times O_2$ and the lifting $\uparrow_{I1\times O1}$ defines it over $I_1 \times I_2 \times U \times V \times O_1 \times O_2$.

Definition 4.5. The synchronous composition of FSMs M_A and M_B yields the FSM $M_A \bullet M_B$ with language $L(M_A) \bullet L(M_B)$. If the language $L(M_A) \bullet L(M_B) = \{\varepsilon\}$ then $M_A \bullet M_B$ is a trivial FSM.

Here we notice that the above definition is sound because the language $L(M_A) \bullet L(M_B)$ is an FSM language if $L(M_A)$ and $L(M_B)$ are FSM languages. The language $L(M_A) \bullet L(M_B)$ may correspond to a complete or partial FSM according to whether the language is I-progressive or not. Then by subset construction and reduction we produce a reduced observable FSM. In summary, we convert from FSMs M_A and M_B to the automata accepting their FSM languages, operate on them and then we convert back from the resulting automaton to an FSM; then we produce a reduced observable FSM.

Supposing now that M_A and the specification M_C of the composition are known and M_B is unknown we define the equation $M_A \bullet M_X \leq M_C$ to capture the FSMs that in place of M_X let the synchronous composition be a reduction of the specification FSM M_C. We solve the equation by building first the related language equation $L(M_A) \bullet L(M_X) \subseteq L(M_C)$ where $L(M_A)$ and $L(M_C)$ are the FSM languages associated to FSMs M_A and M_C. Then we derive the FSM M_S associated to the largest solution S. When there is no ambiguity we will denote by $A \bullet X \subseteq C$ the language equation $L(M_A) \bullet L(M_X) \subseteq L(M_C)$.

Given alphabets I_1, I_2, U, V, O_1, O_2 and FSM M_A over inputs $I_1 \times V$ and outputs $U \times O_1$, and FSM M_C over inputs $I_1 \times I_2$ and outputs $O_1 \times O_2$, let us consider the FSM equation $M_A \bullet M_X \leq M_C$ whose unknown is an FSM over inputs $I_2 \times U$ and outputs $V \times O_2$.

Definition 4.6. FSM M_B is a *solution* to the FSM equation if and only if $M_A \bullet M_B \leq M_C$.

Converted to the related FSM languages, we obtain the associated language equation $L(M_A) \bullet L(M_X) \subseteq L(M_C)$ (or simply $A \bullet X \subseteq C$), where A is an FSM language over $I_1 \times V \times U \times O_1$, C is an FSM language over alphabet $I_1 \times I_2 \times O_1 \times O_2$ and the unknown FSM language is over alphabet $I_2 \times U \times V \times O_2$. We want to characterize the solutions to the latter equation as FSM languages. We know from Theorem 2.1 that the largest solution is the language $\overline{A \bullet \overline{C}}$. When A and C are FSM languages the equation is always solvable, since the language $\overline{A \bullet \overline{C}}$ has the empty string, i.e., the trivial FSM is a solution to the equation. However, in general $S = \overline{A \bullet \overline{C}}$ is not prefix-closed, i.e., in general S is not an FSM language. To compute the largest FSM language contained in S, that is S^{FSM}, we must compute the largest prefix-closed subset of S. Let M_S denote an FSM with the language S^{FSM}, i.e., the largest solution to the FSM equation $M_A \bullet M_X \leq M_C$.

For logic synthesis applications, we assume that M_A and M_C are complete FSMs and we require the solution M_B is a complete FSM too. The latter can be obtained as a complete reduction of the largest complete solution that is the largest complete submachine of the M_S. If M_S has no complete submachine then the equation has no complete solution. The language of the largest complete submachine of M_S is $Prog(S^{FSM})$.

[5] Use the same order $I_1 \times I_2 \times U \times V \times O_1 \times O_2$ in the languages $L(M_A)\uparrow_{I2\times O2}$ and $L(M_B)\uparrow_{I1\times O1}$.

Theorem 4.4. Let A and C be FSM languages. The largest FSM language that is a (complete) solution to the equation is given by S^{FSM} ($Prog(S^{FSM})$) where $S = \overline{A \bullet \overline{C}}$. S^{FSM} contains at least the empty string.

Furthermore, we restrict our attention to the solution B that is compositionally I-progressive, i.e., such that $A_{\uparrow I2\times O2} \cap B_{\uparrow I1\times O1}$ is the language of a complete FSM over inputs $I_1 \times I_2$. Since $A_{\uparrow I2\times O2} \cap B_{\uparrow I1\times O1}$ is prefix-closed and so it corresponds to a partial FSM, we have to restrict it so that it also is I-progressive, which corresponds to a complete FSM. If $A_{\uparrow I2\times O2} \cap (S^{FSM})_{\uparrow I1\times O1}$ is I-progressive then S^{FSM} is the largest compositionally I-progressive solution to the equation. However, not each non-empty subset of S^{FSM} inherits the feature of being compositionally I-progressive. If S^{FSM} is not compositionally I-progressive, then denote by $cProg(S^{FSM})$ the largest compositionally I-progressive subset of S^{FSM}. We start with i=1, $S_1 = S^{FSM}$. Compute $R_i = A_{\uparrow I2\times O2} \cap S_{\uparrow I1\times O1}$. If the language R_i is I-progressive then $cProg(S^{FSM}) = S_i$. Otherwise, obtain $Prog(R_i)$ and compute $L_i = S_i \setminus (R_i \setminus Prog(R_i))_{\downarrow I2\times U\times V\times O2}$. If $L_i{}^{FSM} = \varnothing$ then $cProg(S^{FSM}) = \varnothing$, and the equation has no compositionally I-progressive solution. Otherwise, assign $S_{i+1} = L_i{}^{FSM}$ and increment i by 1. The above procedure always terminates and returns the largest compositionally I-progressive solution (if exists). Here we notice that differently from the largest Moore and complete solutions in general the largest compositionally I-progressive solution is not a submachine of the largest solution to the equation, i.e., is not a submachine of the FSM with the language S^{FSM}. A sufficient condition to insure that $A_{\uparrow I2\times O2} \cap B_{\uparrow I1\times O1}$ is an I-progressive FSM language is that B satisfies the Moore property. If S^{FSM} is Moore then it is the largest Moore solution, and the equation has a compositionally I-progressive solution. However, the equation can have a compositionally I-progressive solution when there is no Moore solution.

The first author was partly supported by the scientific program «Russian Universities». Both the first and the third authors gratefully acknowledge the support of NATO travel grant. The second author gratefully acknowledges the support of the MADESSII Project (Italian National Research Council). The fourth author gratefully acknowledges the support of the NSERC.

REFERENCES

1. *Yevtushenko N., Villa T., Petrenko A., Brayton R.K., Sangiovanni-Vincentelli A.* Solving equations in logic synthesis. – Tomsk: Spectrum Publishers, 1999. – 27 p. (In Russian).
2. *Yevtushenko N., Villa T., Petrenko A., Brayton R.K., Sangiovanni-Vincentelli A.* Logic synthesis by equation solving // Proceedings of XVI Intern. Workshop on Logic synthesis, USA, 2000. – P. 11-14.
3. *Yevtushenko N., Villa T., Petrenko A., Brayton R.K., Sangiovanni-Vincentelli A.* Solving a parallel language equation // Proceedings of the ICCAD'01, USA, 2001. – P. 103-110.
4. *Hopkroft J.E., Ulman J.D.* Introduction to automata theory, Languages and Computations: Addison-Wesley Publishing Company, 1979.
5. *Starke P.H.* Abstract Automata. – New-York: American Elsevier Publishing Company, 1972.