Progressive Solutions to a Parallel Automata Equation

Sergey Buffalov[1], Khaled El-Fakih[2], Nina Yevtushenko[1], and Gregor v. Bochmann[3]

[1] Tomsk State University, Russia
bsa@tric.tomsk.gov.ru, yevtushenko@elefot.tsu.ru

[2] Department of Computer Science, American University of Sharjah, UAE
kelfakih@aus.ac.ae

[3] School of Information Technology and Engineering, University of Ottawa, Canada
bochmann@site.uottawa.ca

Abstract. In this paper, we consider the problem of deriving a component X of a system knowing the behavior of the whole system C and the other components A. The component X is derived by solving the parallel automata equation $A \lozenge X \cong C$. We present algorithms for deriving the largest progressive solution to the equation that combined with A does not block any possible action in C and we introduce a new simulation relation between automata in order to characterize all progressive solutions.

1 Introduction

The *equation solving* problem is to describe a behavior of a component X of a system knowing the specifications of the other components and of the whole system. This problem may be formulated by the equation $A \lozenge X \cong C$ over finite automata, where A represents the specification of the known part of the system, $\lozenge$ is a parallel composition operator, $\cong$ is a trace equivalence relation, and C represents the specification of the whole system.

In 1980, a first paper [2] (see also [7]) gives a solution to the problem for the case when the system behavior is described in terms of labeled transition systems (LTS). This work was later extended to the cases where the behavior of the components is described in CCS or CSP [8], by FSM [9, 14] or input/output automata [11, 6, 3].

The applications of the equation solving problem were first considered in the context of the design of communication protocols, where the components A and C represent two protocol entities [7]. Later it was recognized that this method could also be useful for the design of protocol converters in communication gateways [5, 12, 6], and for the selection of test cases for testing a module in a context [10]. Another application area of equation solving is the design of controllers for discrete event systems [13, 1]. Solutions to the automata equation $A \lozenge X \cong C$ are characterized in [3, 14] as proper reductions of the largest solution. However, not each solution to the equation is of practical use. Usually we are required to get a composed system that does not block any action that is possible in the specification, i.e. we are interested in what is called a *progressive* solution [6]. It is desirable to determine the largest progressive solution [6, 14] to the equation (if it exists) that contains all progressive solutions. In this case, a progressive solution is a reduction of the largest progressive solution. That is, the set of traces of a progressive solution is a subset of that of the largest progressive solution. The largest progressive solution can be viewed as a reservoir of all possible solutions of our interest. However, not each reduction of the largest progressive solution possesses the property. For this reason, in order to determine an optimal solution we need to completely characterize all the reductions of the largest progressive solution that are progressive. The problem has been studied in [3] for Input/Output automata.

In this paper, we generalize the results given in [6]. Given a solution to the automata equation, we suggest a procedure that derives its largest progressive reduction. First, we split the states of the given solution and obtain an equivalent *perfect* automaton. This automaton has a nice property: If a given sequence cannot be in a progressive solution, then this sequence takes the perfect automaton from its initial state to a state that is only reachable through non-progressive sequences. Consequently, we delete from the perfect automaton all states that are only reachable through non-progressive sequences and we obtain the largest progressive reduction of a given solution. Moreover, in this paper, we consider automata with both accepting and non-accepting states and we establish necessary and sufficient conditions for a solution to be progressive. For this purpose, for each state of a perfect automaton, we associate an appropriate collection of regular sets of actions. A solution is progressive if and only if there exists a simulation relation between the solution and the perfect automaton such that for each state pair of the simulation relation, the language generated at the state of the solution intersects each set of the collection associated with the state of the perfect automaton. In Section 3 we present the details of the above described work, while in Section 2 we include all related definitions. In Section 4 we conclude the paper.

2 Finite State Automata Relations, Operators, and Composition

An *alphabet* V is a finite nonempty set of symbols. As usual, we let V^* denote the set of all finite sequences of symbols of V including the *empty sequence* ε. A *language* over the alphabet V is a subset of V^*. Given a sequence $\alpha \in V^*$ and an alphabet W, a *W-restriction* of α, written $\alpha_{\downarrow W}$, is obtained by deleting from α all symbols that belong to the set $V \setminus W$. If α has no letters from alphabet W then the W-restriction $\alpha_{\downarrow W}$ is the empty word. In the paper, we consider only regular languages, i.e. languages that are represented by finite state automata [4].

A *finite state automaton*, often called an *automaton* throughout the paper, is a quintuple $P = \langle S, V, \delta_P, s_0, F_P \rangle$, where S is a finite nonempty set of states with the initial state s_0 and a subset F_P of *final* (or *accepting*) states, V is an alphabet of actions, and $\delta_P \subseteq S \times V \times S$ is a transition relation. We say that there is a transition from a state s to a state s' labeled with an action v, if and only if the triple (s, v, s') is in the transition relation δ_P.

The automaton P is called *deterministic*, if for each state $s \in S$ and any action $v \in V$ there exists at most one state s', such that $(s, v, s') \in \delta_P$. If P is not deterministic, then it is called *nondeterministic*.

As usual, the transition relation δ_P of the automaton P can be extended to sequences over the alphabet V. The extended relation is also denoted by δ_P and is a subset of $S \times V^* \times S$. By definition, for each state $s \in S$ of the automaton P the triple (s, ε, s) is in the relation δ_P. Given a triple $(s, \alpha, s') \in \delta_P$ and an action $v \in V$, the triple $(s, \alpha v, s'')$ belongs to δ_P, if and only if $(s', v, s'') \in \delta_P$.

Given a state s of the automaton P, the set $L^s(P) = \{\alpha \in V^* | (s_0, \alpha, s) \in \delta_P\}$ is called the language *accepted at the state s* and the set $L_s(P) = \{\alpha \in V^* | \exists s' \in F_P\ ((s, \alpha, s') \in \delta_P)\}$ is called the language, *generated at the state s*. The language, generated by the automaton P at the initial state, is called the *language generated* or *accepted by the automaton P* and is denoted by $L(P)$, for short. The language $L(P)$ is the union of all languages accepted at final states of P. By definition, the language $L(P)$ has the empty sequence, if and only if the initial state of P is final. It is worth to note that different automata can accept the same language. When we are interested only in the language of a given automaton we can use a reduced form of the automaton that does not have superfluous states at which the empty language is generated. In this paper, we call an automaton *reduced* if the language generated at each state is not empty[1]

The automaton $\langle \{s_0\}, V, \delta, s_0, \{s_0\} \rangle$, where $\delta = S \times V \times S$, is called *chaos* and is denoted by CHAOS(V). By definition, the chaos automaton CHAOS(V) accepts the language V^*.

A state s of an automaton is said to be *reachable*, if there exists a sequence that takes the automaton from the initial state to s, i.e. the state s accepts a nonempty language. The automaton is called *connected*, if each state is reachable. Any state that is not reachable, can be deleted from the automaton without affecting the language of the automaton. By this reason, we further consider only connected automata.

Here, we recall some relations between finite automata defined over the same alphabet. Let $P = \langle R, V, \delta_P, r_0, F_P \rangle$ and $B = \langle T, V, \delta_B, t_0, F_B \rangle$ be automata.

A relation $\varphi \subseteq T \times R$, is called a *simulation relation* [1], if for each pair $(t, r) \in \varphi$ the following holds:

1. $t \in F_B \Rightarrow r \in F_P$.
2. For each $(t, v, t') \in \delta_B$ there exists $r' \in R$ such that $(r, v, r') \in \delta_P$ and $(t', r') \in \varphi$.

The automaton P *simulates* the automaton B or the automaton B is said to be *simulated* by the automaton P, written $P \sqsupseteq_\varphi B$, if there exists a simulation relation φ, such that the pair (t_0, r_0) of the initial states is in the relation φ.

The state t of the automaton B is called a *reduction* of the state r of automaton P, written $t \le r$, if the language of R generated at state t is a subset of that generated by the P at state r. Automaton R is called a *reduction* of the automaton P, written $R \le P$, if the language of R is a subset of that of P, i.e. $L(R) \subseteq L(P)$. An automaton accepting the empty language is a reduction of any automaton over the same alphabet.

The simulation relation refines the reduction relation [1], i.e. if $P \sqsupseteq_\varphi B$ then $B \le P$. The converse is not always true. However, a deterministic automaton simulates any of its reductions, i.e. the following statement holds.

Proposition 1. [1]. Given a deterministic automaton P, let B be an automaton over the same alphabet. Then B is a reduction of P, if and only if P simulates B. □

[1] A reduced automaton may have equivalent states, i.e. states where equal languages are generated.

Automata P and B are called *trace equivalent* or simply *equivalent*, written $R \cong B$, if they accept the same language, i.e. B is a reduction of P and vice versa.

Let $P = \langle R, V, \delta_P, r_0, F_P \rangle$ and $R = \langle Q, W, \delta_R, q_0, F_R \rangle$ be two automata. We further describe some operations over finite state automata that will be used throughout the paper.

Deterministic representation. Given the automaton P, there exists an equivalent deterministic automaton obtained from P by applying the algorithm of subset construction [4]. We denote this automaton by **DFA**(P) and call it the *deterministic representation* of P. Note that a state of **DFA**(P) is a subset of states of P and accepts the intersection of their languages. Due to the construction, the following statement holds.

Prefix closure. Given the automaton P, the automaton $<P>$ is obtained from P by declaring all states of P as accepting states. The language of the automaton $< P >$ is the prefix closure of that of P, i.e. the language of $< P >$ comprises each prefix of each sequence of the language $L(P)$.

Intersection. If alphabets V and W intersect then the *intersection* $P \cap R$ of automata P and R is the largest connected sub-machine of the automaton $\langle S \times Q, V \cap W, \delta, (s_0, q_0), F_P \times F_R \rangle$. Given an action $a \in V \cap W$ and a state (s, q), there is a transition at the state (s, q) labeled with a, if and only if there are transitions at states s and q labeled with a, i.e. $\delta = \{((s, q), a, (s', q')) \mid (s, a, s') \in \delta_P \wedge (q, a, q') \in \delta_R\}$. The automaton $P \cap R$ accepts the intersection of languages $L(P)$ and $L(R)$. If V and W are disjoint then intersection of P and R is not defined, since the alphabet of an automaton can not be empty.

Proposition 2. Given an automaton P, let $B = \langle T, V, \delta_B, t_0, F_B \rangle$ be a reduced reduction of P and t be a state of B. There exists a state r of P such that the pair (t,r) is a state of the intersection of $B \cap P$. Moreover, if P is deterministic, then for each state (t,r) of the intersection $B \cap P$ state t is a reduction of r. □

Restriction. Given an alphabet U, the *U-restriction* of P is the deterministic form $P{\downarrow_U}$ of the automaton $\langle S, U, \delta, s_0, F_P \rangle$, where $\delta = \{(s, u, s') \mid \exists\, \alpha \in V^* \; \exists\, (s, \alpha, s') \in \delta_P \; (\alpha{\downarrow_U} = u)\}$. The automaton $P{\downarrow_U}$ accepts the language $L(P){\downarrow_U} = \{\alpha \in U^* \mid \exists \beta \in L(P) \; (\alpha = \beta{\downarrow_U})\}$ called the *U-restriction* of the language $L(P)$. The restriction of the language is empty if and only if the language is empty. Due to the definition of the restriction, the following statement can be established.

Proposition 3. Given the U-restriction $P{\downarrow_U}$ of the automaton P and state $p = \{s_1, ..., s_k\}$ of the $P{\downarrow_U}$, let L^p be the language accepted at state p. The set p is the set of all states of P that accept the language over the alphabet V with the U-restriction L^p. □

Expansion. Given an alphabet U, the *U-expansion* of P is the automaton $P{\uparrow_U} = \langle S, V \cup U, \delta, s_0, F_P \rangle$, where $\delta = \delta_P \cup \{(s, u, s) \mid s \in S \wedge u \in U \setminus V\}$. The automaton $P{\uparrow_U}$ is obtained from P by adding at each state a loop transition labeled with each action of the alphabet $U \setminus V$. If U is a subset of V then the automaton $P{\uparrow_U}$ coincides with the automaton P. Automaton $P{\uparrow_U}$ accepts the language $L(P){\uparrow_U} = \{\alpha \in (V \cup U)^* \mid \exists \beta \in L(P) \; (\alpha{\downarrow_V} = \beta)\}$ called the *U-expansion* of the language $L(P)$. The U-expansion of the language is empty if and only if the language is empty.

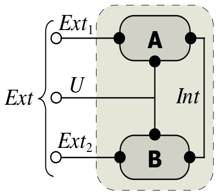


Fig. 1. Automata composition

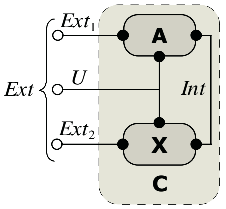


Fig. 2. Equation solving paradigm

Consider a system of two interacting automata $A = \langle T, W, \delta_A, t_0, F_A \rangle$ and $B = \langle S, V, \delta_B, s_0, F_B \rangle$ shown in Figure 1. We assume A and B execute each action of the set $V \cap W$ together when both of them are ready to execute the action. Moreover, automata A and B share actions from the sets $Ext_1 = W \setminus V$ and $Ext_2 = V \setminus W$, respectively, with an environment and execute these actions independently from each other, but not simultaneously. Moreover, we suppose that the subset $U \subseteq V \cap W$ of actions shared by the automata can be observed externally. Thus, actions of the set $Ext = Ext_1 \cup Ext_2 \cup U$ are called *external*, while actions from the alphabet $Int = (V \cap W) \setminus U$ are called *internal*. For an external observer, the automata interaction is described by the sequence of external actions. However, two consecutive external actions can be separated by a sequence of internal actions.

Given the set Ext of external actions, the *composition of automata A* and B is the automaton $A \lozenge_{Ext} B \cong (B{\uparrow_W} \cap A{\uparrow_V}){\downarrow_{Ext}}$. The composition accepts the language $(L(A){\uparrow_W} \cap L(B){\uparrow_V}){\downarrow_{Ext}}$. By definition, if a component automaton accepts the empty language, then the composition accepts the empty language as well.

3 Solving Automata Equations

Let $A = \langle S, W, \delta_A, s_0, F_A \rangle$ and $C = \langle Q, Ext, \delta_C, q_0, F_C \rangle$ be two deterministic reduced automata. A notation $A \lozenge_{Ext} X \cong C$ is called an *equation* w.r.t. a free variable X, which is considered to be an automaton with a given alphabet $V \subseteq W \cup Ext$. One may think of a composition $A \lozenge_{Ext} X$ as a network possessing the desired external behavior C as it is shown in Figure 2. The automaton A is usually called a *context*, and the automaton C is usually called a *specification*. Accepting states of the specification can be viewed as finishing an appropriate task [13].

An automaton B over the alphabet V is called a *solution* to the equation $A \lozenge_{Ext} X \cong C$, if $A \lozenge_{Ext} B \cong C$. Since the set of classes of equivalent automata is ordered by the reduction relation, we can introduce the *largest solution* that includes all solutions to the equation $A \lozenge_{Ext} X \cong C$ as its reductions, i.e. each solution to the equation is a reduction of the largest solution.

Not each solution to the equation is known to be of practical use. A practical solution is required to be progressive. A solution is called *progressive* if, when combined with the context, it will not block the occurrence of an external event if the latter is possible in the specification. If an equation has a progressive solution we will be interested in characterizing all such solutions in order to be able to select an optimal one according to some criteria. In general, the equation may have an infinite number of progressive solutions. Thus, the problem of characterizing all of them appears to be not trivial. In this paper, we further show that if the equation has a progressive solution then it has a largest progressive solution. The set of traces of a progressive solution is a subset of that of the largest progressive solution. Thus, the largest progressive solution can be viewed as a general solution to the equation. Any progressive solution is a reduction of the largest. However, the converse is not true. Therefore, to completely characterize progressive solutions we first want to find the largest progressive solution to the equation $A \lozenge_{Ext} X \cong C$ and then describe all its reductions that are progressive. We note that when the unknown component has no external actions a technique for the derivation of the largest progressive solution is proposed in [6].

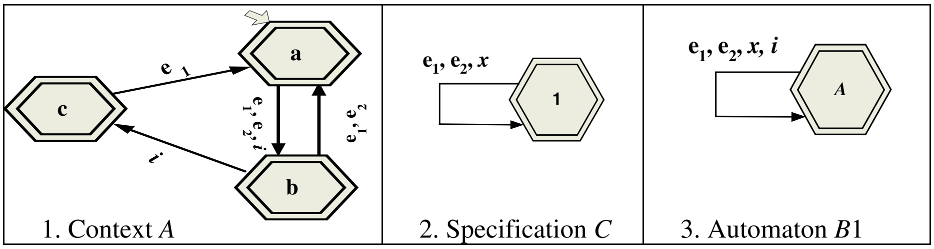


Fig. 3. The example of equation solving

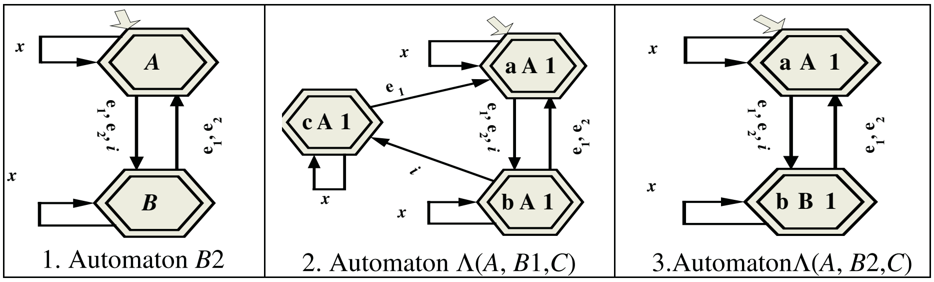


Fig. 4. Progressive and non-progressive solutions

As an example of equation solving (see Figure 3), we consider the context A defined over the alphabet $W = \{e_1, e_2, i\}$, and the specification C defined over the alphabet $Ext = \{e_1,e_2,x\}$ of external actions. Moreover, consider the automaton B_1 defined over the alphabet $V = \{e_1, e_2, i, x\}$ and shown in Figure 3.3. The behavior of the whole system $A \lozenge_{\{e_1,e_2,x\}} B_1$ is equivalent to the given specification. Thus, B_1 is a solution to the equation $A \lozenge_{Ext} X \cong C$.

3.1 A Progressive Solution

In this subsection, we introduce the notion of a progressive solution to the equation $A \lozenge_{Ext} X \cong C$. Then, we give a detailed overview of the method how to derive a largest progressive solution if it exists. Afterwards, we present the consecutive steps of this method along with application examples.

Let $A = \langle S, W, \delta_A, s_0, F_A\rangle$ be a deterministic context, $C = \langle Q, Ext, \delta_C, q_0, Q\rangle$ be a deterministic specification.

Given an automaton $P = \langle R, V, \delta_P, r_0, F_P\rangle$ over alphabet V, a state (s, r, q) of the automaton $A\uparrow_V \cap P\uparrow_W \cap C\uparrow_{W\cup V}$ is called *progressive*, if the *Ext*-restriction of the language generated at the (s, r, q) coincides with the language generated by the specification C at state q, i.e. $L_{(s,r,q)}(A\uparrow_V \cap P\uparrow_W \cap C\uparrow_{W\cup V})\downarrow_{Ext} = L_q(C)$. Otherwise, the state (s, r, q) is called *non-progressive*.

A solution B to the equation $A \lozenge_{Ext} X \cong C$ is called *progressive*, if each state in $A\uparrow_V \cap B\uparrow_W \cap C\uparrow_{W\cup V}$ is progressive. By definition, if B is a progressive solution then B combined with the context does not block an external event that is possible in the specification.

Hereafter, for simplicity of presentation, we let the automaton $\Lambda(A, P, C)$ denote the automaton $A\uparrow_V \cap P\uparrow_W \cap C\uparrow_{W\cup V}$. A state of $\Lambda(A, P, C)$ is a triple (s, r, q), where s is a state of the automaton A, r is a state of the automaton P, and q is a state of the automaton C.

We note that the equation $A \lozenge_{Ext} X \cong C$ can have both progressive and non-progressive solutions. For example, consider the context A shown in Figure 3.1 and the specification C shown in Figure 3.2. The solution B_1 shown in Figure 3.3 is not progressive, since the automaton $\Lambda(A, B_1, C)$, shown in Figure 4.2, has a non-progressive state c1A. On the other hand, another solution B_2, shown in Figure 4.1, is progressive, since all the states of the automaton $\Lambda(A, B_2, C)$, shown in Figure 4.3, are progressive.

Since a progressive solution is defined through properties of the automaton $\Lambda(A, P, C)$, we establish some properties of the states of the automaton and of the states of its V-restriction to the alphabet V of a solution. By definition of the expansion operator, we establish conditions for a triplet (s, r, q) to be reachable from the initial state of the intersection $A\uparrow_V \cap B\uparrow_W \cap C\uparrow_{W\cup V}$.

Proposition 4. Let (s, r, q) be a triplet, where s is a state of the automaton A, r is a state of the automaton P, and q is a state of the automaton C. The triplet is a state of the intersection $A\uparrow_V \cap B\uparrow_W \cap C\uparrow_{W\cup V}$ if and only if the there exists a sequence β over the alphabet $W\cup V$ such that the W-restriction of β takes the context A from the initial state to state q, the V-restriction of β takes the automaton P from the initial state to state r, and the *Ext*-restriction of β takes the specification C from the initial state to state q. □

Proposition 5. Given a deterministic automaton P over the alphabet V, let B be a reduction of P and (t,r) be a state of the intersection $B\cap P$. Given states s and q of A and C, if the triplet (s,t,q) is a state of the intersection $A\uparrow_V \cap B\uparrow_W \cap C\uparrow_{W\cup V}$, then the triplet (s,r,q) is a state of the intersection $A\uparrow_V \cap P\uparrow_W \cap C\uparrow_{W\cup V}$. Moreover, the language generated at state (s,t,q) of the $A\uparrow_V \cap B\uparrow_W \cap C\uparrow_{W\cup V}$ is a subset of that generated at state (s,r,q) of the automaton $A\uparrow_V \cap P\uparrow_W \cap C\uparrow_{W\cup V}$. □

Consider now the state $p = \{(s_1, r_1, q_1), \ldots, (s_k, r_k, q_k)\}$ of the V-restriction of the automaton $A\uparrow_V \cap P\uparrow_W \cap C\uparrow_{W\cup V}$ and let L^p be the language accepted at state p. Due to Proposition 3, the set p is the set of all states of the intersection that accept the language with the V-restriction L^p. Therefore, each state $r_1, \ldots, r_k$ accepts the language

L^P in the automaton P (Proposition 4). Since P is deterministic, there exists only one state accepting the language, i.e. the following statement holds. □

Proposition 6. Given the V-restriction of the automaton $A{\uparrow_V} \cap P{\uparrow_W} \cap C{\uparrow_{W \cup V}}$, let $p = \{(s_1, r_1, q_1), ..., (s_k, r_k, q_k)\}$ be a state of the V-restriction. If P is deterministic then $r_1 = ... = r_k$. □

Given a solution M to the equation $A \Diamond_{Ext} X \cong C$, we now establish necessary and sufficient conditions for a state the of the automaton $\Lambda(A, M, C)$ to be progressive

Given an automaton $A = \langle S, W, \delta_A, s_0, F_A \rangle$ representing the context and an automaton $C = \langle Q, Ext, \delta_C, q_0, Q \rangle$ representing the specification, consider the automaton $M = \langle R, V, \delta_M, r_0, F_M \rangle$. Let (s, r, q) be a state of the automaton $\Lambda(A, M, C)$ and $e \in Ext$ be an external action such that there is a transition from state q with the action e. If the action e takes the automaton C from the state q to a non-final state then we define the set $Re[(s, r, q), e]$ to be the set of sequences $\beta \in (W \cup V)^*$ such that β is a prefix of a sequence in the language generated at state (s, r, q) and $\beta{\downarrow_{Ext}} = e$. If the action e takes the automaton C from the state q to a final state then the set $Re[(s, r, q), e]$ is defined to be the set of sequences $\beta \in (W \cup V)^*$ such that β is in the language of the automaton $\Lambda(A, M, C)$ generated at state (s, r, q) and $\beta{\downarrow_{Ext}} = e$.

Formally, if the action e takes the automaton C from the state q to a non-final state then $Re[(s, r, q), e] = \{\beta \mid \beta{\downarrow_{Ext}} = e \ \& \ \beta \in \ < L_{(s,r,q)}(\Lambda(A, P, C))>\}$. If the action e takes the automaton C from the state q to a final state then $Re[(s, r, q), e] = \{\beta \mid \beta{\downarrow_{Ext}} = e \ \& \ \beta \in L_{(s,r,q)}(\Lambda(A, P, C))\}$.

Based on the construction of the sets $Re[(s, r, q), e]$, we can show, by induction, that each state of the automaton $\Lambda(A, M, C)$ is progressive if and only if for each state (s, r, q) of the automaton and each external action e for which there is a transition from state q with the action e, the set $Re[(s, r, q), e]$ is not empty. The above result is stated in the following proposition.

Proposition 7. Given a solution M to the equation $A \Diamond_{Ext} X \cong C$, each state of the automaton $\Lambda(A, M, C)$ is progressive if and only if for each state (s, r, q) of the automaton and each external action e for which there is a transition from state q with the action e, the set $Re[(s, r, q), e]$ is not empty. □

Here we note that since the restriction and prefix closure of a regular language are regular, each set $Re[(s, r, q), e]$ is a regular set and thus, can be represented as an automaton.

3.2 An Overview of the Method for Solving the Problem

For automata that accept prefix-closed languages, and for a composition topology where the alphabet of a solution is a subset of that of the context, i.e. $V \subseteq W$, it is shown in [6] that the equation $A \Diamond_{Ext} X \cong C$ has a progressive solution, if and only if a largest progressive solution to the equation exists. In Section 3.5 we generalize the result for the more general composition topology shown in Figure 1. Moreover, we propose an algorithm that returns the largest progressive reduction of an automaton over the alphabet V (if exists). As a corollary, we obtain a largest progressive solution if we start from the chaos automaton CHAOS(V) or the largest solution to the equation. The obtained largest progressive solution has a nice property that it includes all progressive solutions, and thus can be used later for the characterization of these solutions. We also note that each automaton P over the alphabet V can be considered as an automaton over a bigger alphabet $V'\supseteq V$ with the same set of transitions and final states. For this reason, if the equation $A \Diamond_{Ext} X \cong C$ has no progressive solution over the alphabet $W \cup Ext$ then the equation has no progressive solution over any alphabet $V \subset W \cup Ext$. Therefore, without any loss of generality we only consider the largest progressive solution over the alphabet $V = W \cup Ext$.

The problem of characterizing all progressive reductions of a largest progressive solution is not trivial, since the number of these reductions is infinite and, moreover, not each reduction of a largest progressive solution is progressive. We solve the problem by introducing the notion of a perfect automaton, the operation of a regular extension, and regular simulation relation. The idea behind the approach is described in details in Section 3.4.

3.3 Largest Progressive Solutions

Given two progressive solutions to the equation $A \Diamond_{Ext} X \cong C$, their union is also a progressive solution. In other words, the set of all progressive solutions ordered by the reduction relation is a semilattice. In general, the semilattice of all progressive solutions can be infinite, and thus the problem of determining if it has a largest element, i.e. if the equation $A \Diamond_{Ext} X \cong C$ has a largest progressive solution, is not trivial.

It is known [6] [14] that a solvable equation has a largest solution, i.e. there exists an automaton M, such that any solution to the equation is a reduction of M. Therefore, before we determine a largest progressive solution, we could check if the equation is solvable at all. Given a context $A = \langle S, W, \delta_A, s_0, F_A\rangle$ and a specification $C = \langle Q, Ext, \delta_C, q_0, Q\rangle$ where each state is final, in [7] it is shown that the equation $A \Diamond_{Ext} X \cong C$ is solvable if and only if the automaton $A\Diamond C \backslash A\Diamond \overline{C}$ [2] is a solution to the equation. The automaton $A\Diamond C \backslash A\Diamond \overline{C}$ is not the largest solution to the equation since the language of this automaton does not comprise traces that do not participate in the composition with the given context. The largest solution to an arbitrary automata equation is the automaton $M \cong \overline{A \Diamond_{Ext} \overline{C}}$ [YVPBS99] if the composition $A \Diamond_{Ext} M$ is equivalent to C. If the composition is not equivalent to C, then the equation has no solution. The automaton B_1 shown in Figure 3.3 is the chaos machine over alphabet $\{e_1, e_2, i, x\}$ and therefore, B_1 is the largest solution to the equation $A \Diamond_{Ext} X \cong C$, where A and C are shown in Figures 3.1 and 3.2, respectively.

If the largest solution $M = \langle R, V, \delta_M, r_0, F_M\rangle$ to the equation $A \Diamond_{Ext} X \cong C$ exists and is progressive, then it is a largest progressive solution. Otherwise, in order to obtain a largest progressive solution or to show that it does not exist, we first build a so-called *perfect* automaton $M_{perfect}$ and then we use this automaton to get the largest reduction that is a progressive solution to the equation. The perfect automaton can also be used for the complete characterization of all progressive solutions of a given automata

[2] $\overline{C}$ denotes the automaton that accepts the complement of the language of C.

equation. In the following two subsections we give the ideas and the details of building a perfect automaton and its largest progressive reduction.

Let $P = \langle R, V, \delta_P, r_0, F_P \rangle$ be an automaton over the alphabet V. Given a non-progressive state (s, r, q) of the automaton $\Lambda(A, P, C)$, let β be a sequence that takes this automaton from its initial state to the state (s, r, q). We call the corresponding sequence $\alpha \in L^r(P)$, where α is the V-restriction of β (i.e. $\alpha = \beta_{\downarrow V}$), a *non-progressive sequence*. In other words, if a sequence $\alpha \in L^r(P)$ is non-progressive then there is no progressive solution with this sequence. Otherwise, the sequence α is called *progressive*. That is for a progressive sequence α, each sequence β with the V-restriction α must take the automaton $\Lambda(A, P, C)$ from the initial state to a progressive state.

In order to derive the largest reduction of P that is a progressive solution (if it exists) we have to eliminate all non-progressive sequences from the language of the P. However, in general, the number of non-progressive sequences is infinite. On the other hand, we cannot delete states from the automaton P, since a state of the automaton P can accept both progressive and non-progressive sequences. As an example, consider the context A shown in Figure 3.1, the specification C shown in Figure 3.2, and the largest solution B_1 shown in Figure 3.3. The initial state A of B_1 accepts a non-progressive sequence *ii*. However, if we delete this state, this implies that there is no progressive solution. Nevertheless, the solution B_2 (Figure 4.1) is progressive. i.e. there exists a progressive solution. Consequently, we refine the automaton P and obtain the equivalent automaton P_{perfect} such that the largest reduction is the largest sub-machine of P_{perfect} that does not have non-progressive states. The number of sub-machines of an automaton is finite and the largest sub-machine without non-progressive states can be obtained by iterative deleting non-progressive states. Therefore, we refine P by splitting its states in order to obtain an equivalent automaton P_{perfect} such that each state of P_{perfect} accepts either progressive or non-progressive sequences. The obtained automaton P_{perfect} is called *perfect* (w.r.t. the given context and specification). Each sub-machine obtained from P_{perfect} by deleting from P_{perfect} one of its non-progressive states, with its incoming and outgoing transitions, is perfect too. Thus, we (iteratively) delete from P_{perfect} all states that accept non-progressive sequences. If the initial state of P_{perfect} is deleted, then we conclude that there is no progressive solution to the given equation that is a reduction of the P. Otherwise, the obtained automaton is the largest reduction of P that is a progressive solution. A largest progressive solution can be obtained when the initial automaton P is the largest solution to the equation or it is the chaos automaton over alphabet V, i.e. CHAOS(V), that generates the language V^*. Any automaton over alphabet V is a reduction of CHAOS(V).

3.3.1 Perfect Automata

In this subsection, we first discuss the idea of constructing a perfect automaton. Afterwards, we present an algorithm for building such an automaton.

The problem we face with an automaton P over alphabet V that in particular can be seen as a largest solution to the equation, is the following. Given a state r of P, there could exist sequences β_1 and β_2 in the language of $\Lambda(A, P, C)$ with the V-restrictions α_1 and α_2 such that the sequence β_1 takes the automaton $\Lambda(A, P, C)$ from the initial state to the triplet (s_1, r, q_1) that is progressive, while the β_2 takes the automaton from the initial state to the triplet (s_2, r, q_2) that is non-progressive. Therefore, sequences α_1 and α_2 accepted at the state r of the automaton P possess different features: sequence α_1 is progressive while α_2 is a non-progressive sequence. Our objective is to delete the state r from P because it is reachable through the non-progressive sequence α_2. However, when deriving the largest progressive reduction of the P we cannot delete r, since it is also reachable through the sequence α_1 that can be executed by a progressive solution. Consequently, for such two sequences, we would like to split r into several states and obtain a perfect automaton P_{perfect} such that these sequences take P_{perfect} to two different states. As a result, we will not have any triplet in $\Lambda(A, P_{\text{perfect}}, C)$ that is reachable by two sequences such that the V-restriction of these sequences are equal to α_1 and α_2. Therefore, each state r' of P_{perfect} will only be reachable through either progressive or non-progressive sequences. This allows us later to delete state r' if it accepts non-progressive sequences without losing a progressive solution that is a reduction of the automaton P. Formally, given the context A and the specification C, an automaton $P = \langle R, V, \delta_P, r_0, F_P\rangle$ is called *perfect* (w.r.t. the given context and specification), if for any state (s, r, q) of the automaton $\Lambda(A, P, C)$ the V-restriction of the language accepted at the state (s, r, q) coincides with the language accepted at the state r of the automaton P, i.e. $L^r(P) = L^{(s, r, q)}(\Lambda(A, P, C))_{\downarrow V}$. We further let P_{perfect} denote the perfect automaton obtained from P.

Proposition 8. Given a perfect automaton P_{perfect} (w.r.t. a given context A and specification C), each state of P_{perfect} accepts either non-progressive or progressive sequences. Moreover, each sub-machine, $Sub(P_{\text{perfect}})$, obtained from P_{perfect} by deleting from P_{perfect} one of its non-progressive states with its incoming and outgoing transitions, is also perfect. □

Given an automaton P over the alphabet V, the idea of constructing a perfect automaton P_{perfect} that is equivalent to P is as follows. For each sequence α in the language of the automaton P, we determine the subset of all triplets in $\Lambda(A, P, C)$ reachable through sequences with the V-restriction equal to α. In general, for many sequences in the language of P, we will have the same subset of triplets in $\Lambda(A, P, C)$. Each triplet of a subset accepts the language of sequences with the same V-restriction (Proposition 3). Consider states s and q of the automata A and C such that the triplet (s, r, q) is a state of the subset. Due to Proposition 4, the V-restriction of the intersection $L^s(A) \cap L^{(s, r, q)} \cap L^q(C)$ where $L^s(A)$ and $L^q(C)$ are languages accepted at states s and q, equals to $L^{(s, r, q)}$. The latter implies that such subsets of triplets can serve as states of the automaton P_{perfect}. Consequently, since each triplet of the subset accepts the language with one and the same set of V-restrictions, then the V-restrictions of the sequences accepted by the triplets are either progressive or non-progressive, depending if the subset includes a non-progressive triplet. We then add to the language of P_{perfect}, all sequences of the language of P that do not participate in the composition with the context A. This is done in order for P_{perfect} to be equivalent to P. For example, the automaton B_1 shown in Figure 3.3 is not perfect. The progressive sequence e_1 and the non-progressive sequence $e_1 i$ take the automaton B_1 to the same state A. On the contrary, the automaton B_2 shown in Figure 4.1 is perfect. Here we note that for every automaton P, there exists an equivalent perfect automaton $P_{\text{perfect.}}$ Below we propose an algorithm to derive P_{perfect}.

Algorithm 1. Deriving the perfect automaton P_{perfect} of P

Input: The automaton $P = \langle R, V, \delta_P, r_0, F_P\rangle$, context $A = \langle S, W, \delta_A, s_0, F_A\rangle$, and specification $C = \langle Q, Ext, \delta_C, q_0, F_C\rangle$.
Output: The deterministic perfect automaton P_{perfect} that is equivalent to P.

Declare all states of $A{\uparrow_V}$ and $C{\uparrow_{W\cup V}}$ as accepting states and derive the automaton $\Lambda(A, P, C) = A{\uparrow_V} \cap P{\uparrow_W} \cap C{\uparrow_{W\cup V}}$. Restrict the intersection to the alphabet V and let $P' = \langle R', V, \delta_{P'}, r'_0, F_{P'}\rangle$ denote the resulting automaton. The automaton $P_{\text{perfect}} = \langle R \cup R', V, \delta, r'_0, F_{P'} \cup F_P\rangle$ is obtained from P and P' as follows. The initial state of P_{perfect} is the initial state of P', and the transition relation δ contains the union of the transition relations $\delta_{P'}$ and δ_P of both automata P' and P. Moreover, for each transition (r,a,r') of P, we add a transition with the label a from the subset of triplets of P' comprising the triplet (s, r, q) to state r' of P if and only if there is no transition with the label a from the subset.

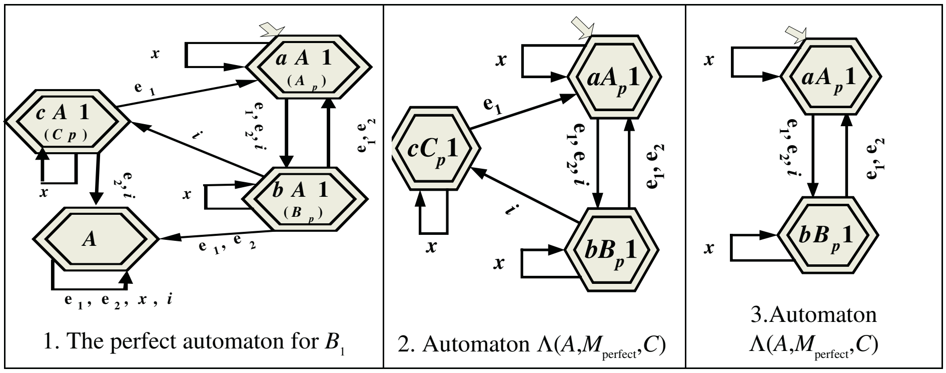
Theorem 1. The automaton P_{perfect} returned by Algorithm 1 is deterministic, equivalent to P and perfect (w.r.t. the given context and specification). €

3.3.2 An Algorithm for Deriving a Largest Progressive Solution

Given the equation $A \lozenge_{Ext} X \cong C$, let P be an automaton over the alphabet V. We let P_{perfect} denote the perfect automaton for P. Below, we present an algorithm that trims P_{perfect} by iteratively deleting its states that correspond to non-progressive states in $\Lambda(A, P_{\text{perfect}}, C)$. If the initial state is deleted in P_{perfect}, then none of the reductions of P is a progressive solution to the $A \lozenge_{Ext} X \cong C$. Otherwise, the obtained submachine $P_{\textit{Largest-Prog.}}$ of P_{perfect} is the largest progressive reduction of P.

Algorithm 2. Deriving a largest progressive solution

Input: The automaton $P = \langle R, V, \delta_P, r_0, F_P\rangle$, context $A = \langle S, W, \delta_A, s_0, F_A\rangle$, and specification $C = \langle Q, Ext, \delta_C, q_0, F_C\rangle$.
Output: The largest reduction $P_{\textit{Largest-Prog.}}$ of P that is a progressive solution to the equation $A \lozenge_{Ext} X \cong C$ if it exists.
Step-1. Derive, using Algorithm 1, the perfect deterministic automaton P_{perfect} that is equivalent to the P and the automaton $\Lambda(A, P_{\text{perfect}}, C)$.
Step-2.If a state (s, r, q) of the automaton $\Lambda(A, P_{\text{perfect}}, C)$ is non-progressive then:
-Delete from the automaton P_{perfect} the state r and all the states that become unreachable from the initial state;
-Delete from the automaton $\Lambda(A, P_{\text{perfect}}, C)$ each state that has the second component equal to r and all the states that become unreachable from the initial state;
-Repeat **Step 2** of the algorithm until no more states can be deleted in the automaton $\Lambda(A, P_{\text{perfect}}, C)$. If the initial state is deleted then the equation $A \lozenge_{Ext} X \cong C$ has no progressive solutions that are reductions of P. Otherwise, the obtained submachine $P_{\textit{Largest-Prog.}}$ of P_{perfect} is the largest reduction of P that is a progressive solution to the equation $A \lozenge_{Ext} X \cong C$.



1. The perfect automaton for B_1 2. Automaton $\Lambda(A,M_{\text{perfect}},C)$ 3.Automaton $\Lambda(A,M_{\text{perfect}},C)$

Fig. 5. Deriving a largest progressive solution.

If we start with the chaos machine CHAOS(V) (i.e. P = CHAOS(V)) or with the largest solution M (i.e. $P = M$) to the equation $A \Diamond_{Ext} X \cong C$, Algorithm 2 returns a largest progressive solution to the equation if it exists.

As an application example of Algorithm 2, we consider the context A and the specification C shown in Figures 3.1 and 3.2, respectively. We assume that $\{e_1, e_2, i, x\}$ is the alphabet of a solution X to the equation $A \Diamond_{\{e_1, e_2, x\}} X \cong C$. The largest solution $M = B_1$ to this equation shown in Figure 3.3 is not progressive. Therefore, first, we apply **Step 1** of the above algorithm to M and we derive the perfect deterministic automaton M_{perfect} shown in Figure 5.1. For the sake of simplicity, we rename the states of M_{perfect} as shown in Figure 5.1, where $A_p = aA1$, $B_p = bA1$, $C_p = cA1$. The automaton $\Lambda(A, M_{\text{perfect}}, C)$ is shown in Figure 5.2. State cC_p1 of this automaton is not progressive. Consequently, in **Step 2** we delete from $\Lambda(A, M_{\text{perfect}}, C)$ all the states that include state c, i.e. cC_p1, and we obtain the automaton shown in Figure 5.3. Moreover, we delete state C_p from the perfect automaton M_{perfect}. Since the automaton in Figure 5.3 does not have non-progressive states, the algorithm terminates and returns the automaton shown in Figure 4.1 as the largest progressive solution.

3.4 Characterization of Progressive Solutions and Re-simulation Relation

A characterization of progressive solutions to the equation $A \Diamond_{Ext} X \cong C$ over I/O automata is proposed in [3]. In this section, we introduce a new *Re*-simulation relation between finite automata that allows us to describe all progressive solutions for parallel automata equations. That is we propose a complete characterization of all progressive solutions of the automata equation $A \Diamond_{Ext} X \cong C$.

Our characterization is very close to that proposed in [3]. We associate with each state of a largest progressive solution a family of regular sets. A reduction of the largest progressive solution is a progressive solution if and only if it is simulated by the largest progressive solution and for each pair (t,r) of the simulation relation, the language of the reduction at state t intersects each regular set of the family associated with the state r of the largest progressive solution.

We recall that if the largest solution to a given automata equation is progressive then it is a largest progressive solution. Moreover, we note that we may have many largest progressive solutions to a given equation. Unfortunately, not each largest progressive solution can be used for the complete characterization of all progressive solutions. However, the largest progressive solution that is perfect can be used for the complete characterization of all progressive solutions.

Given an arbitrary largest progressive solution M_L, each progressive solution is a reduction of M_L. However, not each reduction of M_L is progressive. By definition, a reduction $B=\langle T, V, \delta_B, t_0, F_B\rangle$ of M_L is progressive if and only if each state of the automaton $\Lambda(A,B,C)$ is progressive. Given a state (s,t,q) of $\Lambda(A,B,C)$, let $\beta\in L^{(s,t,q)}(\Lambda(A,B,C))$ be a sequence such that the V-restriction α of β (i.e. $\alpha=\beta_{\downarrow V}$) takes the intersection $B\cap M_L$ of automata M_L and B to state pair (t,r). Since B is a reduction of the deterministic M_L, for a given t there always exists such r. Now, let (s,r,q) be a state of the automaton $\Lambda(A,M_L,C)$ that is reachable from the initial state through the sequence β. Since M_L is progressive, each state (s,r,q) of the automaton $\Lambda(A,M_L,C)$ is progressive. This means that for each external action e that can be executed at state q of the specification C the set of sequences $Re[(s,r,q),e]$ is not empty (Proposition 7) and thus, the V–restriction $Re[(s,r,q),e]_{\downarrow V}$ of $Re[(s,r,q),e]$ is also not empty. Since B is a reduction of M_L, the set of sequences generated at state (s,t,q) of $\Lambda(A,B,C)$ is a subset of that generated at state (s,r,q) of $\Lambda(A,M_L,C)$ (Proposition 5). Therefore, for (s,t,q) to be progressive, at least one sequence from the set $Re[(s,r,q),e]_{\downarrow V}$ should be generated at state t.

Therefore, for each state (s,r,q) and for each external action e produced at state q, we associate the corresponding set of sequences $Re[(s,r,q),e]_{\downarrow V}$ with state r of M_L. We call the automaton M_L with associated sets of sequences a *regular extension* of M_L, and we denote it by $\mathbf{Re}(M_L)$.

Therefore, for a reduction B of M_L to be progressive, it is sufficient that for each pair (r,t) in the intersection $M_L\cap B$, the automaton B at state t generates at least one sequence from each set of sequences associated with r. In this case, we say that M_L *Re-simulates B*.

Unfortunately, for an arbitrary largest progressive solution that is not a perfect automaton, we could have a progressive solution that does not satisfy the above condition. This happens when for some (s,r,q) of the automaton $\Lambda(A,M_L,C)$ there does not exist a corresponding state (s,t,q) in $\Lambda(A,B,C)$. However, by construction, we have selected un-needed (or superfluous) sequences from all the sets that relate to (s,r,q) and every external action e that can be produced at state (s,r,q), independent of whether there exists a (s,t,q). Actually, for this reason, we could have progressive solutions that are not Re-simulated by M_L. However, the above cannot happen for a perfect automaton due to Theorem 1 and according to the following proposition.

Proposition 9. Given the context A and specification C, let $M_{\text{perfect}}=\langle R, V, \delta_M, r_0, F_M\rangle$ be a deterministic perfect automaton that is equivalent to a largest progressive solution M_L of the equation $A\Diamond_{Ext}X\cong C$. Let the automaton B be a reduction of M_{perfect}, a pair (t,r) be a state of the intersection $B\cap M_{\text{perfect}}$, and states s and q be states of the automata A and C. If the automaton $\Lambda(A,M_{\text{perfect}},C)$ has a state (s,r,q) then the automaton $\Lambda(A,B,C)$ has a state (s,t,q). □

Consequently, in order to have a complete characterization of all progressive solutions over a given alphabet, we use a largest progressive solution $M_{Perfect}$ that is a deterministic perfect automaton. The automaton M_{perfect} has the nice property that each progressive solution is Re-simulated by it and vice versa.

By definition, if a triple (s, r, q) is not a state of the automaton $\Lambda(A, M_{\text{perfect}}, C)$, then for each external action $e \in Ext$ the set $Re[(s, r, q), e]$ is empty. We denote by $Re(r)$ the set of all nonempty sets $Re[(s, r, q), e]\downarrow_V$, for all $(s, q, e) \in S \times Q \times Ext$.

Given an automaton M_{perfect}, we let the pair $\langle M_{\text{perfect}}, Re\rangle$ denote the regular extension of the automaton M_{perfect}, where Re is a collection of all $Re(r)$, where r is a state of M_{perfect}.

An automaton $B = \langle T, V, \delta_B, t_0, F_B\rangle$ is said to be *simulated* by the regular extension of an automaton $M = \langle R, V, \delta_M, r_0, F_M\rangle$, if B is simulated by M with the simulation relation $\varphi \subseteq T \times R$, such that for each pair $(t, r) \in \varphi$ the prefix-closure of the language generated at state t of the automaton B intersects each set from $Re(r)$, i.e. the following holds: $Re(r) \neq \varnothing \Rightarrow \forall L \in Re(r)\ (L \cap L_t(B) \neq \varnothing)$.

The simulation relation φ with the above property is called a *regular simulation relation* or simply a *Re-simulation relation.*

Let $A = \langle S, W, \delta_A, s_0, F_A\rangle$ be the context and $C = \langle Q, Ext, \delta_C, q_0, Q\rangle$ be a reduced deterministic specification. Due to Theorem 1, if there exists a progressive solution to the equation $A \Diamond_{Ext} X \cong C$ then there exists a perfect automaton M_{perfect} such that $L^r(M_{\text{perfect}}) = (L^{(s,r,q)}(\Lambda(A, M_{\text{perfect}}, C)))\downarrow_V$. M_{perfect} can be derived as proposed in Algorithm 1.

Theorem 2. Given a deterministic perfect largest progressive solution M_{perfect} to the equation $A \Diamond_{Ext} X \cong C$, a reduction B of the automaton M_{perfect} is a progressive solution to the equation if and only if B is Re-simulated by M_{perfect}. €

4 Conclusions

In this paper we address the problem of characterizing progressive solutions to a composed automata equation where the automata communicate by rendezvous. A progressive solution is of special interest, since when combined with the context it does not block any action of the environment that is possible according to the specification. Particularly, we have proposed a technique for deriving the largest reduction of an automaton that is a progressive solution to the equation (if it exists). The technique can be used in order to determine the largest progressive solution. However, not each reduction of the largest progressive solution is progressive and therefore, the problem of characterizing all progressive solutions is not trivial. In order to solve the problem, we have introduced a new Re-simulation relation between finite automata that allows us to describe all progressive solutions. The complete characterization of progressive solutions enables us to select an "optimal" solution, where an optimal solution can be defined as the one with the least number of states, actions and transitions, or the fastest one. Currently, we are working on techniques for deriving optimal progressive solutions.

Acknowledgments

The first and the third authors acknowledge the partial support of the program "Russian Universities". The second author acknowledges the partial support of the American University of Sharjah.

References

[1] Barrett G., Lafortune S.: Bisimulation: The Supervisory Control Problem, and Strong Model Matching for Finite State Machines. Discrete Event Dynamic Systems: Theory and Application. 8(4):377-429 (1998).
[2] Bochmann G. v., Merlin, P.: On the Construction of Communication Protocols. ICCC, 1980, 371-378, reprinted in "Communication Protocol Modeling", edited by C. Sunshine, Artech House Publ. (1981).
[3] Drissi J., Bochmann G. v.: Submodule Construction for Systems of I/O Automata. ftp://beethoven.site.uottawa.ca/Publications/Dris99b.pdf
[4] Hopcroft J. E., Ullman J. D. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979).
[5] Kelekar, S.G.H: Synthesis of Protocols and Protocol Converters Using the Submodule Construction Approach. Proc. PSTV, XIII, A. Danthine et al (Eds) (1994).
[6] Kumar R., Nelvagal S., Marcus S. I.: A Discrete Event Systems Approach for Protocol Conversion. Discrete Event Dynamical Systems: Theory and Applications. 7(3):295-315 (1997).
[7] Merlin P., Bochmann G. v.: On the Construction of Submodule Specifications and Communication Protocols. ACM Trans. On Programming Languages and Systems. 5(1):1-25 (1983).
[8] Parrow, J.: Submodule Construction as Equation Solving in CCS. Theoretical Computer Science. Vol. 68 (1989).
[9] Petrenko, A., Yevtushenko, N.: Solving Asynchronous Equations. In Proc. of IFIP FORTE/PSTV'98 Conf., Paris, Chapman-Hall (1998).
[10] Petrenko, A., Yevtushenko, N., Bochmann, G. v., Dssouli, R.: Testing in Context: Framework and Test Derivation. Computer Communications Journal, Special issue on Protocol engineering. Vol. 19, 1236-1249 (1996).
[11] Qin, H., Lewis, P.: Factorisation of Finite State machines Under Strong and Observational Equivalences. Journal of Formal Aspects of Computing, Vol. 3, 284-307 (1991).
[12] Tao, Z., Bochmann, G.v., Dssouli, R.: A Formal Method for Synthesizing Optimized Protocol Converters and its Application to Mobile Data Networks. Mobile Networks & Applications. 2(3):259-69 (1997).
[13] Wonham W. M., Ramadge P. J.: On the Supremal Controllable Sublanguage of a Given Language. SIAM J. Control. Optim. 25(3):637-659 (1987).
[14] Yevtushenko, N., Villa, T., Brayton, R.K., Petrenko, A., Sangiovanni-Vincentelli, A.: Solving a Parallel Language Equation. Proc. of the ICCAD'01, USA, (2001).