Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 362 (2006) 17–32
Theoretical Computer Science
www.elsevier.com/locate/tcs

Progressive solutions to a parallel automata equation

Khaled El-Fakih[a,*], Nina Yevtushenko[b], Sergey Buffalov[b], Gregor v. Bochmann[c]

[a] *Department of Computer Science, American University of Sharjah, P.O. Box 26666, Sharjah, United Arab Emirates*
[b] *Tomsk State University, 36 Lenin Str., Tomsk 634050, Russia*
[c] *School of Information Technology and Engineering, University of Ottawa, 800 King Edward Ave, P.O. Box 450, Stn A, Ottawa, Ont., Canada K1N 6N5*

Received 4 February 2006; accepted 4 May 2006

Communicated by Z. Esik

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 $A \diamond X \cong C$. We present an algorithm for deriving a largest progressive solution to the equation that combined with A does not block any possible action in C and we establish conditions that allow us to characterize all progressive solutions.

Keywords: Equation solving; Parallel automata equations; 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 the specification of the whole system. This problem may be formulated in the form of an equation $A \diamond X \cong C$ over finite automata, where A represents the specification of the known part of the system, $\diamond$ 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 [9]) gives a solution to the problem for the case where 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 [10], by FSM [11,16] or input/output automata [4,8,13].

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 [9]. Later it was recognized that this method could also be useful for the design of protocol converters in communication gateways [7,8,14], and for the selection of test cases for testing a module in a context [5,12]. Another application area of equation solving is the design of controllers for discrete event systems [1,15]. Solutions to the automata equation $A \diamond X \cong C$ are characterized in [4,16,17] as proper

* Corresponding author. Tel.: +971 6 5152556; fax: +971 6 5152979.
E-mail addresses: kelfakih@aus.edu (K. El-Fakih), yevtushenko@elefot.tsu.ru (N. Yevtushenko), buf@mail2000.ru (S. Buffalov), bochmann@site.uottawa.ca (G.v. Bochmann).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.05.034

reductions of a largest solution, i.e., the language of each solution is a subset of that of a 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 according to its specification, i.e., we are interested in what is called a *progressive* solution [8]. If an equation has a progressive solution then the equation is known to have a largest progressive solution [8,17] that contains all progressive solutions. In this case, a progressive solution is a reduction of a largest progressive solution. We note that a largest progressive solution to an equation is not unique; however, any two largest progressive solutions are equivalent, i.e., any two largest progressive solutions accept one and the same language. A largest progressive solution can be viewed as a reservoir of all possible solutions of our interest. However, not each reduction of a largest progressive solution is progressive. For this reason, in order to determine an optimal solution we need to completely characterize all the reductions of a largest progressive solution that are progressive. The problem has been studied in [4] for input/output (I/O) automata.

In this paper, similar to [8] we solve the problem for finite automata where the context A and specification C are deterministic. In particular, we generalize the results obtained in [8] in four important directions. First, we consider a more general topology where a component of interest X may have external actions that are not shared with the known part (context) A. In this case, we cannot use the approach proposed in [8], since the language of C is not the external restriction of a subset of the language of the known part A. For this reason, we use another approach for equation solving [17] where a largest solution to the equation is explicitly derived. Second, we present an algorithm that derives a largest progressive reduction (if it exists) of a given automaton. This is helpful when an equation is solved over special automata, for example, over Finite State Machines (FSMs), where the set of actions is partitioned into two disjoint sets of inputs and outputs and where in interaction sequences each input is followed by an output. In this case, a solution has to be an FSM, i.e., a reduction of a maximum FSM. We note that our algorithm returns a largest progressive solution if we start with a largest solution to the equation or with a maximum automaton, which represents the set of all sequences over a given alphabet. Third, in this paper, we consider automata with both accepting and non-accepting states. Fourth, we establish necessary and sufficient conditions for a reduction of a largest progressive solution to be progressive. For this purpose, for each state of a largest progressive reduction, we associate an appropriate collection of regular sets of actions. A solution is progressive if and only if for each state pair of the intersection of the solution and a largest progressive solution, the language generated at the state of the solution intersects each set of the collection associated with the state of a largest progressive solution.

A preliminary version of this work appeared in [3]. Here we extend [3] by formalizing the progressiveness notion, refining the proposed algorithm for deriving a largest progressive reduction of a given automaton by deleting the superfluous step of deriving the largest solution to the equation, and characterizing progressive solutions without the introduction of a simulation relation.

This paper is organized as follows. Section 2 includes preliminary definitions and results while Section 3 covers additional concepts and the major results. Section 4 concludes the paper.

2. Preliminaries

2.1. Finite state automata

An *alphabet* V is a finite non-empty set of symbols. A finite sequence of symbols of the alphabet is called a *trace* or a *word*. 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 this paper, we consider only regular languages, i.e., languages that are represented by finite state automata [6].

A *finite state automaton*, often called an *automaton* throughout the paper, is a quintuple $A = \langle S, V, \delta_A, s_0, F_A\rangle$, where S is a finite non-empty set of states with the initial state s_0 and a subset F_A of *final* (or *accepting*) states, V is an alphabet of actions, and $\delta_A \subseteq S \times V \times S$ 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_A. An automaton $\langle S', V, \delta'_A, s_0, F'_A\rangle$ is a *submachine* of the automaton A if $S' \subseteq S$, $\delta'_A \subseteq \delta_A$, and $F'_A \subseteq F_A$. Given state s of the automaton A, the *largest submachine* of A that does not contain s is obtained by deleting from A the state s with its incoming and outgoing transitions.

The automaton A 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 $(s, v, s') \in \delta_A$. If A is not deterministic, then it is called *nondeterministic*.

As usual, the transition relation δ_A of the automaton A can be extended to sequences over the alphabet V. The extended relation is also denoted by δ_A and is a subset of $S \times V^* \times S$. By definition, for each state $s \in S$ of the automaton A the triple (s, ε, s) is in the relation δ_A. Given a triple $(s, \alpha, s') \in \delta_A$ and an action $v \in V$, the triple $(s, \alpha v, s'')$ belongs to δ_A, if and only if $(s', v, s'') \in \delta_A$. In this paper, we consider only initially connected automata, i.e., automata where each state is reachable from the initial state. If an automaton has states that are not reachable from the initial state then we consider the *largest connected submachine* of the automaton, i.e., the largest submachine without non-reachable states.

Given a state s of the automaton A, the set $L^s(A) = \{\alpha \in V^* \mid (s_0, \alpha, s) \in \delta_A\}$ is called the language *accepted at the state s* and the set $L_s(A) = \{\alpha \in V^* \mid \exists s' \in F_A((s, \alpha, s') \in \delta_A)\}$ is called the language *generated at the state s*. The language generated by the automaton A at the initial state is called the *language generated* or *accepted by the automaton A* and is denoted by $L(A)$, for short. The language $L(A)$ is the union of all languages accepted at final states of A. By definition, the language $L(A)$ includes the empty sequence, if and only if the initial state of A is final. It is worth noting that different automata may accept the same language. When we are interested only in the language of a given automaton we can use a trim form of the automaton that does not have superfluous states at which the empty language is generated. In this paper, we call an automaton *trim* if the automaton is initially connected and the language generated at each state other than the initial state is not empty. The empty language can be generated at the initial state of a trim automaton if the automaton represents the empty language.

The automaton $\langle\{s_0\}, V, \delta, s_0, \{s_0\}\rangle$, where $\delta = s_0 \times V \times s_0$, is called *maximum* automaton over V and is denoted by $MAX(V)$. By construction, the automaton $MAX(V)$ accepts the language V^*.

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

Proposition 1. *Given a deterministic automaton P, let B be a trim reduction of P and t be a state of automaton B. If the sequence $\alpha \in L^t(B)$ takes the automaton B from the initial state to state t then α takes the automaton P from the initial state to a state r such that t is a reduction of r.*

Proof. Given a sequence $\beta \in L_t(B)$, $\alpha\beta \in L(P)$ because of $B \leqslant P$, and thus, there exists state r that is reached in P through α. Since $P = \langle S, V, \delta_P, s_0, F_P\rangle$ is deterministic, the state r is the only state such that $(s_0, \alpha, r) \in \delta_P$ and therefore, $\beta \in L_r(P)$, i.e., $L_t(B) \subseteq L_r(P)$. □

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

2.2. Operators over finite state automata

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

Deterministic equivalent: Given an automaton $P = \langle R, V, \delta_P, r_0, F_P\rangle$, there exists an equivalent deterministic automaton obtained from P by applying the algorithm of subset construction [6]. In order to derive such deterministic automaton we first consider the automaton $D = \langle Z, V, \delta_D, \{r_0\}, F_D\rangle$, where Z is the set of all non-empty subsets of P and F_D comprises each subset (or state) in Z that includes a final state of P. Given states $z, z' \subseteq R$ of the automaton D and $a \in V$, the triplet $(z, a, z') \in \delta_D$ if and only if $z' = \{r' \mid \exists r \in z\ ((r, a, r') \in \delta_P)\}$. We call the largest connected submachine of the automaton D the *deterministic representation* $\mathbf{D_{FA}}(P)$ of P.

Prefix closure: Given a trim automaton P, the automaton $\langle P\rangle$ is obtained from P by declaring all states of P as accepting states. The language of the automaton $\langle P\rangle$ is the prefix closure of the language accepted by P, i.e., the language of $\langle P\rangle$ comprises each prefix of each sequence of the language $L(P)$. A trim deterministic automaton accepts a prefix-closed language if and only if all its states are accepting.

Intersection: If alphabets V and W intersect then the *intersection* $P \cap C$ of automata P and C is the largest connected submachine of the automaton $\langle R \times Q, V \cap W, \delta, (s_0, q_0), F_P \times F_C\rangle$. Given an action $a \in V \cap W$ and a state (r, q), there is a transition at the state (r, q) labeled with a and leading to state (r', q'), if and only if there are transitions at states r and q labeled with a and leading to states r' and q', respectively, i.e., $\delta = \{((r, q), a, (r', q')) \mid (r, a, r') \in \delta_P \wedge (q, a, q') \in \delta_C\}$. The automaton $P \cap R$ accepts the intersection of languages $L(P)$ and $L(C)$. If V and W are disjoint then the intersection of P and C is not defined.

Proposition 2. *Given an automaton P, let B with the state set T be a trim 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.*

Proof. Due to the definition of the intersection operator, the pair (t, r) is a state of the intersection of $B \cap P$ if and only if there exists a sequence that takes the automaton B to state t while taking the automaton P to state r. Therefore, the second statement of the proposition is a corollary to Proposition 1. Consider now a sequence α that takes the automaton B from the initial state to state t. Since B is trim, there exists a sequence $\beta \in L_t(B)$, i.e., $\alpha\beta \in L(B)$, and thus, $\alpha\beta \in L(P)$ because B is a reduction of P. Therefore, there exists a state r where the sequence α takes the automaton P from the initial state, i.e., the pair (t, r) is a state of the intersection of $B \cap P$. □

Restriction: Given an alphabet U, the *U-restriction* of $P = \langle R, V, \delta_P, r_0, F_P\rangle$, written $P_{\downarrow U}$, is the deterministic equivalent $\mathbf{DFA}(B)$ of the automaton $B = \langle R, U, \delta_B, r_0, F_B\rangle$, where $F_B = F_P$ and $\delta_B = \{(r, u, r') \mid \exists \alpha \in V^*(\exists(r, \alpha, r') \in \delta_P \& (\alpha_{\downarrow U} = u))\}$. 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.

By definition of the restriction operator, the following proposition holds.

Proposition 3. *Given the U-restriction $P_{\downarrow U} = \langle Z, U, \delta, s_0, F\rangle$ of the automaton P and state $z = \{r_1, \ldots, r_k\}$ of $P_{\downarrow U}$, let L^z be the language accepted at state z. The set z has a state r of P if and only if the U-restriction of the language $L^r(P)$ accepted at state r contains L^z.*

Proof. The proof is divided into two parts. In the first part, we link the languages of the automaton $B = \langle R, U, \delta_B, r_0, F_B\rangle$, where $F_B = F_P$ and $\delta_B = \{(r, u, r') \mid \exists \alpha \in V^*(\exists(r, \alpha, r') \in \delta_P \& (\alpha_{\downarrow U} = u))\}$), and its deterministic equivalent $\mathbf{DFA}(B) = P_{\downarrow U}$ and show that state z of the automaton $P_{\downarrow U}$ that is the subset of states of the automaton B, has a state r of B if and only if the language $L^r(B)$ accepted at state r contains $L^z(P_{\downarrow U})$. In the second part, we show that given a state $r \in R$ of automaton P, the U-restriction of the language $L^r(P)$ of the automaton P and the language $L^r(B)$ of the automaton B that are accepted at the state r can differ only with the empty sequence and despite this difference, the statement of the first part holds also for the case when the language $L^r(B)$ is replaced with the U-restriction of the language $L^r(P)$.

Part 1: Given the deterministic representation $\mathbf{DFA}(B) = \langle Z, V, \delta, \{r_0\}, F\rangle$ of the automaton B we show that a state z of the automaton $\mathbf{DFA}(B)$ has a state r of B if and only if the language $L^r(B)$ accepted at state r contains $L^z(\mathbf{DFA}(B))$. Consider states r and z of the automata B and $\mathbf{DFA}(B)$, and a sequence γ over alphabet U. By induction on the length of sequence γ, it can be shown that $\gamma \in L^z(\mathbf{DFA}(B)) \Leftrightarrow \forall r \in z\ (\gamma \in L^r(B)) \& \forall r \notin z\ (\gamma \notin L^r(B))$.

If γ is the empty sequence ε then, by definition, $\varepsilon \in L^z\mathbf{DFA}(B)) \Leftrightarrow z = z_0$ and $\varepsilon \in L^r(B) \Leftrightarrow r = r_0$. By construction of $\mathbf{DFA}(B)$, $z_0 = \{r_0\}$ and thus, $\varepsilon \in L^z(\mathbf{DFA}(B)) \Leftrightarrow \forall r \in z\ (\varepsilon \in L^r(B)) \& \forall r \notin z\ (\varepsilon \notin L^r(B))$. Thus, the basis step is established.

We now assume that the statement holds for each word over alphabet U that has at most k symbols. Consider a word β that has k symbols and $u \in U$. Due to the induction assumption, given states r and z of the automata B and $\mathbf{DFA}(B)$, $\beta \in L^z(\mathbf{DFA}(B)) \Leftrightarrow \forall r \in z\ (\beta \in L^r(B)) \& \forall r \notin z\ (\beta \notin L^r(B))$. By definition, given state y of the automaton $\mathbf{DFA}(B)$, $\beta u \in L^y \Leftrightarrow (z, u, y) \in \delta$. On the other hand, $y = \{n \mid \exists r \in z\ ((r, u, n) \in \delta_B)\}$ and $\beta \in L^r(B) \Leftrightarrow r \in z$. Thus, $\beta u \in L^n(B) \Leftrightarrow n \in y$.

Consider state z of the automaton $\mathbf{DFA}(B)$ and the language $L^z(\mathbf{DFA}(B))$ accepted at state z. We have shown that given a sequence $\gamma \in U^*$, it holds that $\gamma \in L^z(\mathbf{DFA}(B)) \Leftrightarrow \forall r \in z\ (\gamma \in L^r(B)) \& \forall r \notin z\ (\gamma \notin L^r(B))$. Therefore, it holds that $\forall r \in z\ (L^z(\mathbf{DFA}(B)) \subseteq L^r(B))$, while $\forall r \notin z\ (L^z(\mathbf{DFA}(B)) \cap L^r(B) = \emptyset)$. In other words, $r \in z \Leftrightarrow L^z(\mathbf{DFA}(B)) \subseteq L^r(B)$.

Keeping in mind that $\mathbf{DFA}(B) = P_{\downarrow U}$ we obtain the following statement: state z of the automaton $P_{\downarrow U}$ has a state r of B if and only if the language $L^r(B)$ accepted at state r contains $L^z(P_{\downarrow U})$.

Part 2: In this part, we first show that given a state $r \in R$, the U-restriction of the language $L^r(P)$ of the automaton P and the language $L^r(B)$ of the automaton B that are accepted at the state r can differ only with the empty sequence. In other words, we prove that for each state $r \in R$ it holds that $L^r(B) = L^r(P)_{\downarrow U}$, if $r = r_0$ or r cannot be reached in P from the initial state via a sequence over the set $(V \backslash U)^*$. If $r \neq r_0$ and r is reachable in P from the initial state via a sequence over the set $(V \backslash U)^*$ then $L^r(B) = L^r(P)_{\downarrow U} \backslash \{\varepsilon\}$.

In order to prove that the languages $L^r(B)$ and $L^r(P)_{\downarrow U}$ can differ only with the empty sequence, by induction on the length of a non-empty sequence $\gamma \in U^*$ we show that for each state r of the automata B and P, it holds that $\gamma \in L^r(B) \Leftrightarrow \gamma \in L^r(P)_{\downarrow U}$. The basis step is easy. By construction of the automaton B, given $u \in U$, the triplet $(r_0, u, r) \in \delta_B$ if and only if $\exists \alpha \in V^*(\exists (r_0, u, r) \in \delta_P \& (\alpha_{\downarrow U} = u))\}$. Thus, $u \in L^r(B)$ if and only if $u \in L^r(P)_{\downarrow U}$. We now assume that the statement holds for each word over alphabet V that has at most k symbols. Consider a word β that has k symbols and $u \in U$. Due to the induction assumption, given state r of the automata B and P, $\beta \in L^r(B)$ if and only if $\beta \in L^r(P)_{\downarrow U}$. By definition of the automaton B, given state n of B, $\beta u \in L^n(B) \Leftrightarrow \exists r \in R(\beta \in L^r(B) \& \exists \alpha \in V^*((r, \alpha, n) \in \delta_P \& (\alpha_{\downarrow U} = u)))$. Thus, $\beta u \in L^n(B) \Leftrightarrow \exists \chi \in V^*((r, \chi, n) \in \delta_P \& (\chi_{\downarrow U} = \beta u))$, i.e., $\beta u \in L^n(B) \Leftrightarrow \beta u \in L^n(P)_{\downarrow U}$.

Consider now the empty sequence ε. If $r = r_0$ then by definition, both languages $L^r(B)$ and $L^r(P)_{\downarrow U}$ have the sequence ε and according to the statement, $L^r(B) = L^r(P)_{\downarrow U}$. If $r \neq r_0$ then the language $L^r(B)$ does not have the empty sequence. In the case when r cannot be reached in P from the initial state via a sequence over the set $(V \backslash U)^*$, the language $L^r(P)_{\downarrow U}$ also does not have the empty sequence, and thus, the languages $L^r(B)$ and $L^r(P)_{\downarrow U}$ coincide. In the case when r can be reached in P from the initial state via a sequence over the set $(V \backslash U)^*$, the language $L^r(P)_{\downarrow U}$ has the empty sequence that is not in the language $L^r(B)$ and thus, $L^r(B) = L^r(P)_{\downarrow U} \backslash \{\varepsilon\}$.

We now show that despite that the languages $L^r(B)$ and $L^r(P)_{\downarrow U}$ may differ by the empty sequence, the statement of the first part also holds for the case when the language $L^r(B)$ is replaced with the U-restriction of the language $L^r(P)$, i.e., state z of the automaton $P_{\downarrow U}$ has a state r of P if and only if the U-restriction of the language $L^r(P)$ accepted at state r contains $L^z(P_{\downarrow U})$.

By definition of $P_{\downarrow U}$, the only state that accepts the empty sequence ε is the initial state. Thus, given state z of $P_{\downarrow U}$ we consider two cases: $z = z_0$ and $z \neq z_0$. In the former case $z = z_0 = \{r_0\}$ and r_0 is the only state such that the language $L^r(B)$ accepted at state $r = r_0$ contains $L^z(P_{\downarrow U})$ (Part 1), i.e., state $z = z_0$ of the automaton $P_{\downarrow U}$ has a state r of the automaton P if and only if the language $L^r(P) = L^r(B)$ accepted at state $r = r_0$ contains $L^z(P_{\downarrow U})$.

Consider now state z of $P_{\downarrow U}$, $z \neq z_0$. The state z does not accept the empty sequence and the state z has a state r of B if and only if the language $L^r(B)$ accepted at state r contains $L^z(P_{\downarrow U})$ (Part 1). Since the languages $L^r(B)$ and $L^r(P)_{\downarrow U}$ can only differ with the empty sequence that is not in the language $L^z(P_{\downarrow U})$, the state z has a state r of P if and only if the U-restriction $L^r(P)_{\downarrow U}$ of language $L^r(P)$ accepted at state r contains $L^z(P_{\downarrow U})$. □

Expansion: Given an alphabet U, the *U-expansion* of P is the automaton $P_{\uparrow U} = \langle R, V \cup U, \delta, r_0, F_P \rangle$, where $\delta = \delta_P \cup \{(r, u, r) \mid r \in R \,\&\, u \in U \backslash V\}$. The automaton $P_{\uparrow U}$ is obtained from P by adding at each state a loop transition for 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. If $U \cap V = \emptyset$ then the *U-expansion* of a trim automaton that accepts only the empty sequence is a trim maximum automaton over the alphabet U.

2.3. Composition of finite state automata

Consider a system of two interacting automata $A = \langle S, W, \delta_A, s_0, F_A \rangle$ and $B = \langle T, V, \delta_B, t_0, F_B \rangle$ as shown in Fig. 1. We assume that 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 (execute together) actions from the sets $Ext_1 = W \backslash V$ and $Ext_2 = V \backslash W$, respectively, with the 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 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 $Int = (V \cap W) \backslash U$ are called *internal*.

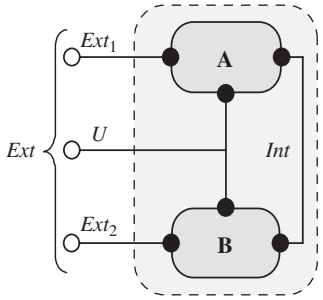


Fig. 1. Automata composition.

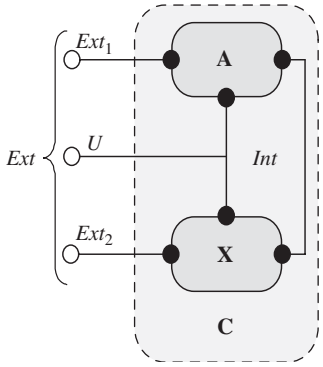


Fig. 2. Equation solving paradigm.

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 $A \diamond_{Ext} B \cong (B_{\uparrow W} \cap A_{\uparrow V})_{\downarrow Ext}$ [16]. 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

3.1. 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 trim deterministic automata. An expression "$A \diamond_{Ext} X \cong C$" is called an *equation* w.r.t. a free variable X that represents an automaton over a given alphabet $V \subseteq W \cup Ext$. The unknown X should capture a behavior that, when composed with A, supports the desired external behavior C, as shown in Fig. 2. The automaton A is usually called the *context*, and the automaton C is usually called the *specification*. Each accepting state of the specification can be viewed as finishing a corresponding task [15].

An automaton B over the alphabet V is called a *solution* to the equation $A \diamond_{Ext} X \cong C$, if $A \diamond_{Ext} B \cong C$. As usual, an equation can have no solution. However, if an equation is solvable then the equation has a largest solution [16]. A solution to the equation $A \diamond_{Ext} X \cong C$ is called a *largest solution* if it includes all solutions as reductions, i.e., each solution to the equation is a reduction of a largest solution. A largest solution to the equation is not unique; however, any two largest solutions are equivalent, i.e., any two largest solutions accept one and the same language. For example, a largest solution to the equation $A \diamond_{Ext} X \cong C$ can be obtained as the automaton $M = \overline{A \diamond_{Ext} \overline{C}}$ [1] [16] if the composition $A \diamond_{Ext} M$ is equivalent to C. If the composition $A \diamond_{Ext} M$ is not equivalent to C, then the equation

[1] Here $\overline{C}$ denotes a complement of C, that is, an automaton that accepts the complement of the language accepted by the automaton C.

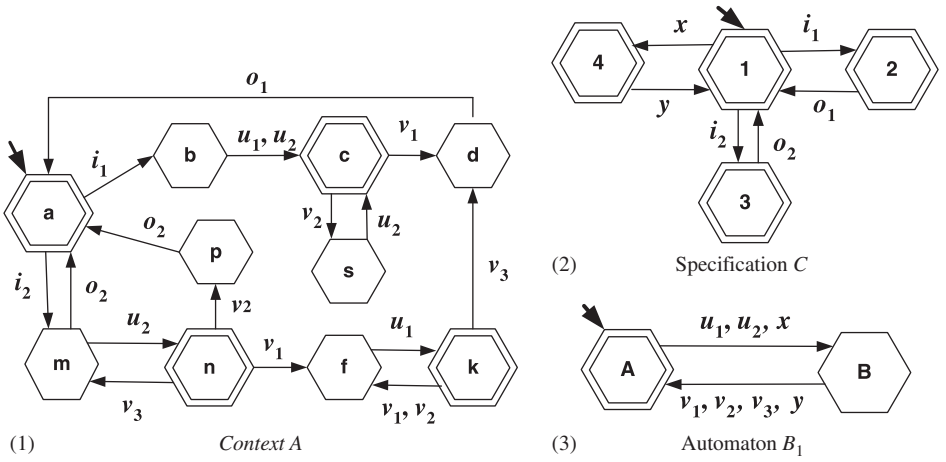


Fig. 3. Equation solving example.

has no solution. Given an automaton P over alphabet V, the intersection $P \cap M$ is a largest reduction of P that can be a solution to the equation.

Not each solution to the equation is of practical use. A practical solution is required to be progressive. A solution is called *progressive* if when combined with the context it does not block (see Section 3.2 below) 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; each progressive solution is a reduction of a largest progressive solution. We note that a largest progressive solution to an equation is not unique; however, any two largest progressive solutions are equivalent, i.e., any two largest progressive solutions accept one and the same language. The set of traces of a progressive solution is a subset of that of a largest progressive solution. Thus, a largest progressive solution can be viewed as a general progressive solution to the equation. Any progressive solution is a reduction of a largest. However, not each reduction of a largest solution is a progressive solution. Therefore, to completely characterize progressive solutions we first want to find a largest progressive solution to the equation $A \diamond_{Ext} X \cong C$ and then describe all its reductions that are progressive solutions.

As an example of equation solving (see Fig. 3), we consider the context A defined over the alphabet $W = \{i_1, i_2, u_1, u_2, v_1, v_2, v_3, o_1, o_2\}$, and the specification C defined over the alphabet $Ext = \{i_1, i_2, o_1, o_2, x, y\}$ of external actions. We assume that $V = \{u_1, u_2, v_1, v_2, v_3, x, y\}$ is the alphabet of a solution X to the equation $A \diamond_{\{i1,i2,o1,o2,x,y\}} X \cong C$. The automaton B_1, defined over the alphabet $V = \{u_1, u_2, v_1, v_2, v_3, x, y\}$ and shown in Fig. 3(3), is a solution to the equation $A \diamond_{Ext} X \cong C$ since the behavior of the whole system $A \diamond_{\{i1,i2,o1,o2,x,y\}} B_1$ is equivalent to the given specification C.

3.2. A progressive solution

In this section, we formally define the notion of a progressive solution to the equation $A \diamond_{Ext} X \cong C$. Let $A = \langle S, W, \delta_A, s_0, F_A \rangle$ be a trim deterministic context, and $C = \langle Q, Ext, \delta_C, q_0, F_C \rangle$ be a trim deterministic specification where $W = Ext_1 \cup Int$ and $Ext = Ext_1 \cup Ext_2 \cup U$.

Given an automaton $P = \langle R, V, \delta_P, r_0, F_P \rangle$, where $V = Ext_2 \cup Int$, an action e of the specification C is *blocked* at state q of C if the external restriction of the language generated at some state (s, r, q) of $A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$ does not contain words with the prefix e. Correspondingly, a state (s, r, q) of $A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$ is called *progressive* if the external restriction of the language generated at state (s, r, q) of $A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$ equals the language generated at state q of the specification C. If P is a solution to the equation $A \diamond_{Ext} X \cong C$ then P is called a *progressive solution*, if each state of $A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$ is progressive. That is for each state (s, r, q) of $A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$, it holds that $L_{(s,r,q)}(A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V})_{\downarrow Ext} = L_q(C)$. By definition, if P is a progressive solution then P combined with

the context does not block an external event that is possible in the specification. Hereafter, for the simplicity of the 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.

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

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 $A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$ if and only if 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 s, 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 trim 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 $A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$.*

Proof. The first statement of the above proposition is a direct corollary to Proposition 4. By the definition of the intersection operator, the language generated at state (s, t, q) of $A_{\uparrow V} \cap B_{\uparrow W} \cap C_{\uparrow W \cup V}$ is the intersection $L_s(A)_{\uparrow V} \cap L_t(B)_{\uparrow W} \cap L_q(C)_{\uparrow W \cup V}$. The intersection of deterministic automata is deterministic; therefore, Proposition 2 implies the second statement of the proposition. □

Consider now the state $z = \{(s_1, r_1, q_1), \ldots, (s_k, r_k, q_k)\}$ of the V-restriction of the automaton $A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$ and let L^z be the language accepted at state z. Due to Proposition 3, the V-restriction of the language accepted at each state of the set z contains L^z. Therefore, each state $r_1, \ldots, r_k$ accepts each sequence of the language L^z in the automaton P (Proposition 4). Since P is deterministic, for each sequence of the language L^z, there exists only one state in P accepting the sequence, i.e., the following statement holds.

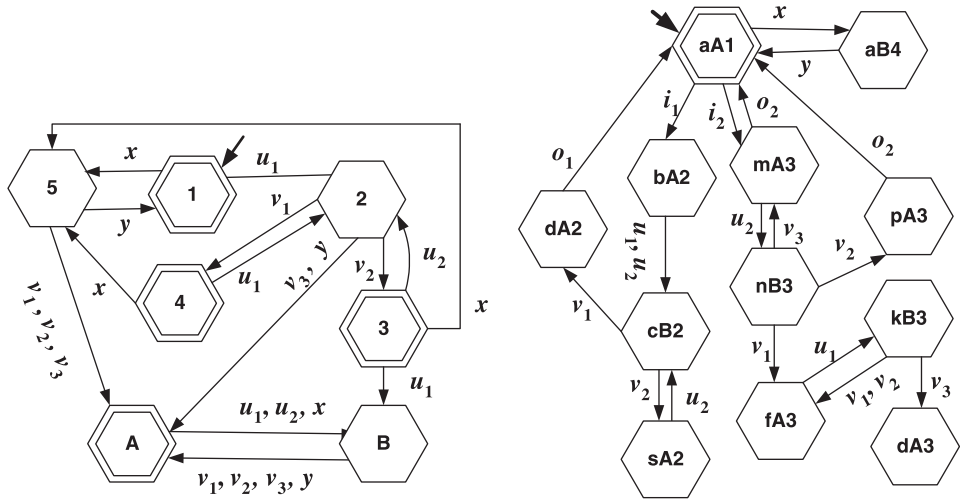
Proposition 6. *Given the V-restriction of the automaton $A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$, let $z = \{(s_1, r_1, q_1), \ldots, (s_k, r_k, q_k)\}$ be a state of this V-restriction. If P is deterministic then $r_1 = \cdots = r_k$.*

Given a trim deterministic automaton $A = \langle S, W, \delta_A, s_0, F_A\rangle$ representing the context and a trim deterministic automaton $C = \langle Q, Ext, \delta_C, q_0, F_C\rangle$ representing the specification, and a trim deterministic solution $M = \langle R, V, \delta_M, r_0, F_M\rangle$ to the equation $A \diamond_{Ext} X \cong C$, we now establish necessary and sufficient conditions for a state of the automaton $\Lambda(A, M, C)$ to be progressive.

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 $Re[(s, r, q), e]$ as 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 as 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 $Re[(s, r, q), e] = \{\beta \mid \beta_{\downarrow Ext} = e \,\&\, \beta \in L_{(s,r,q)}(\langle\Lambda\ (A, M, C)\rangle)\}$. If the action e takes the automaton C from the state q to a final state then $Re[(s, r, q), e] = \{\beta \mid \beta_{\downarrow Ext} = e \,\&\, \beta \in L_{(s,r,q)}(\Lambda\ (A, M, C))\}$. 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 by an automaton.

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



(1) Automaton B_2 (2) Automaton $\Lambda(A,B_1,C)$

Fig. 4. Progressive solution B_2 and the automaton $\Lambda\ (A, B_1, C)$.

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.*

As an example of progressive and non-progressive solutions, consider the context A, the specification C, and the solution B_1 shown in Figs. 3(1)–3(3), respectively, and the automaton $\Lambda\ (A, B_1, C)$ shown in Fig. 4(2). The automaton B_1 is not progressive since the set $Re[(\text{dA3}), o_2]$ is empty. On the other hand, the equation has a progressive solution that is a reduction of B_1. By direct inspection, one can assure that the automaton B_2 shown in Fig. 4(1) is a reduction of B_1 and it is a progressive solution to the equation, since all the states of the automaton $\Lambda\ (A, B_2, C)$ are progressive. We note that a progressive reduction of the automaton B_1 cannot be derived by deleting the non-progressive states of the automaton $\Lambda\ (A, B_1, C)$ shown in Fig. 4(2). If we delete the non-progressive states fA3, kB3, and dA3 and restrict the obtained automaton to the alphabet of the solution, we obtain a non-progressive solution.

3.3. Largest progressive reductions

In this section, we assume that a given automaton P is a solution (not necessarily a largest, and not necessarily progressive) to the equation $A \diamond_{Ext} X \cong C$. We are then interested in finding a largest reduction of P that is a progressive solution of the equation. For this purpose, we first refine the automaton P, i.e., transform P into an equivalent *perfect* automaton P_{perfect}. A largest progressive reduction of P is a submachine of P_{perfect}. This submachine can be obtained by trimming states and transitions from P_{perfect}. The perfect automaton can also be used for the complete characterization of all reductions of P that are progressive solutions to the equation.

In the following two sections we give the ideas and the details of building a perfect automaton and its largest progressive submachine.

3.3.1. Perfect automata

In this section, we first discuss the idea of a perfect automaton and then present an algorithm for building such an automaton.

Let $P = \langle R, V, \delta_P, r_0, F_P\rangle$ be an automaton over the alphabet V. If P is not a progressive solution to the equation, then we have to delete some sequences from the language of P. These are the sequences whose extensions to W take

the composition $\Lambda\ (A, P, C)$ to non-progressive states. However, the number of such sequences can be infinite and in order to have a procedure that terminates, we may try to delete appropriate states of P. Accordingly, all the sequences accepted at these states are deleted from the language of P. However, the problem we face with this procedure is the following. Given a state r of P, the V-restriction of the language accepted at state (s, r, q) of $\Lambda\ (A, P, C)$ is a subset of the language accepted at state r, i.e., $L^{(s,r,q)}{}_{\downarrow V} \subseteq L^r$ (Proposition 4). If state (s, r, q) is not progressive, then each sequence of the set $L^{(s,r,q)}{}_{\downarrow V}$ has to be deleted from the language of the largest progressive reduction of P, since all these sequences take the composition to the state that is not progressive. All the sequences of the set $L^{(s,r,q)}{}_{\downarrow V}$ take the automaton P to state r. But, when the language $L^{(s,r,q)}{}_{\downarrow V}$ is a proper subset of L^r, i.e., when $L^{(s,r,q)}{}_{\downarrow V} \subset L^r$, we cannot delete state r from P since the language accepted at state r may have sequences that can be included in a progressive solution. Those are the sequences that are in $L^r \backslash L^{(s,r,q)}{}_{\downarrow V}$. Therefore, we refine the automaton P and obtain an equivalent perfect (w.r.t. the given context and specification) automaton P_{perfect}. Formally, the automaton P is *perfect* (w.r.t. the given context and specification) if for each state r of P, the V-restriction of the language accepted at state (s, r, q) of $\Lambda\ (A, P, C)$ equals the language accepted at state r, i.e., $L^{(s,r,q)}{}_{\downarrow V} = L^r$. If P is perfect and state r is not progressive, we can delete r without losing any progressive solution that is a reduction of P.

The idea of constructing such a "perfect" automaton P_{perfect} is as follows. For each sequence α in the language of the automaton P, we determine the set 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 set 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 $L^s(A)_{\uparrow V} \cap L^{(s,r,q)}{}_{\uparrow W} \cap L^q(C)_{\uparrow W \cup V}$ where $L^s(A)$ and $L^q(C)$ are languages accepted at states s and q, equals $L^r(P_{\text{perfect}})$. This implies that such subsets of triplets can serve as states of the automaton P_{perfect}. 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 to keep P_{perfect} equivalent to P.

For example, the automaton B_1 shown in Fig. 3(3) is not perfect. The language accepted at state sA2 (Fig. 4(2)) has no sequence $u_1 v_3$ that is accepted at the state A of the automaton B_1. On the contrary, the automaton B_2 shown in Fig. 4(1) is perfect.

Given a deterministic automaton P, the following is an algorithm for deriving P_{perfect}.

Algorithm 1. Deriving the perfect (w.r.t to the given context and specification) automaton P_{perfect} of P.

Input: A trim deterministic automaton $P = \langle R, V, \delta_P, r_0, F_P\rangle$, a trim deterministic context $A = \langle S, W, \delta_A, s_0, F_A\rangle$, and a trim deterministic specification $C = \langle Q, Ext, \delta_C, q_0, F_C\rangle$.

Output: The deterministic perfect (w.r.t. the given context and specification) automaton P_{perfect} that is equivalent to P.

Step 1: Declare all states of $A_{\uparrow V}$ and $C_{\uparrow W \cup V}$ as accepting states and derive the automaton $\Lambda\ (A, P, C) = A_{\uparrow V} \cap P_{\uparrow W} \cap C_{\uparrow W \cup V}$.

Step 2: Restrict the intersection $\Lambda\ (A, P, C)$ to the alphabet V and let $P_{\text{temp}} = \langle R_{\text{temp}}, V, \delta_{\text{temp}}, r_{t0}, F_{\text{temp}}\rangle$ denote the resulting automaton.

Step 3: Add to the language of P_{temp} all sequences of the language of P that do not participate in the composition with the context A. Formally, the automaton $P_{\text{perfect}} = \langle Z, V, \delta_{\text{perfect}}, r_{t0}, F_{\text{temp}} \cup F_P\rangle$ where $Z = R_{\text{temp}} \cup R$, is obtained from P and P_{temp} as follows. The initial state of P_{perfect} is the initial state of P_{temp} and the transition relation δ_{perfect} contains the union of the transition relations δ_{temp} and δ_P of both automata P_{temp} and P. Moreover, for each transition (r_1, a, r_2) of P and each subset z of triplets of P_{temp} that has a triplet (s, r_1, q) with state r_1, we check if there is a transition from the subset with the label a in P_{temp}. If there is no such transition then we add to P_{temp} a transition (z, a, r_2) . Denote P_{perfect} the obtained automaton.

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*).

Proof. By definition of the restriction and intersection operators, P_{temp} is deterministic. Due to Proposition 4, the language of P_{temp} is a subset of that of P. In Step 3, when deriving P_{perfect} from the automaton P_{temp}, the languages of the automata P_{perfect} and P are made equal; thus, P_{perfect} is deterministic and equivalent to P.

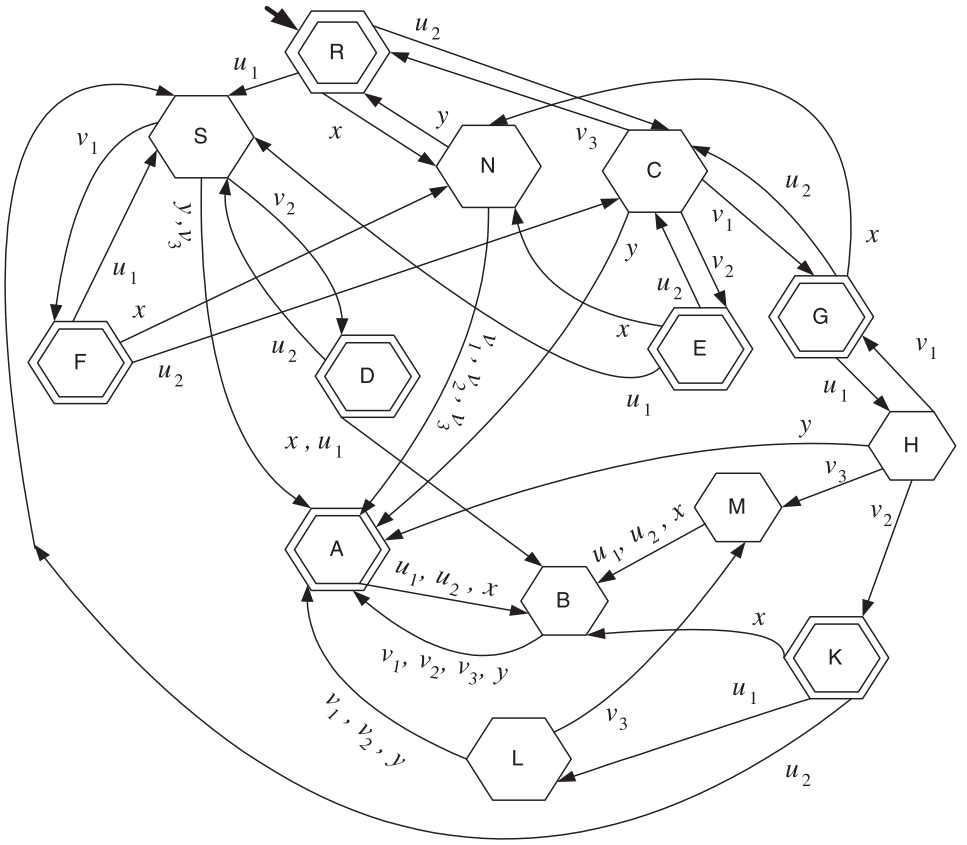


Fig. 5. The perfect automaton M_{perfect}.

In order to show that the obtained automaton P_{perfect} is perfect, we first notice that only sequences of the language of P_{temp} participate in the composition. Consider a state z of the automaton P_{temp} that is the set of triplets $\{(s_1, r, q_1), \ldots, (s_k, r, q_k)\}$ (Proposition 6). The language $L^z(P_{\text{perfect}})$ accepted at state z is the intersection of the V-restrictions of the languages accepted at states (s_j, r, q_j) over all $j = 1, \ldots, k$ (Proposition 3), i.e., formally $L^z(P_{\text{perfect}}) = \bigcap_{j=1,\ldots,k} (L^r(P)_{\uparrow W} \cap L^{s_j}(A)_{\uparrow V} \cap L^{q_j}(C)_{\uparrow W\cup V})_{\downarrow V}$, i.e., for each $j = 1, \ldots, k$ it holds that L^{q_j}:

$$L^z(P_{\text{perfect}}) = (L^z(P_{\text{perfect}})_{\uparrow W} \cap L^{s_j}(A)_{\uparrow V} \cap L^{q_j}(C)_{\uparrow W\cup V})_{\downarrow V}. \tag{1}$$

Given states s and q of the context A and the specification C, the triplet (s, z, q) is a state of the intersection $A_{\uparrow V} \cap (P_{\text{perfect}})_{\uparrow W} \cap C_{\uparrow W\cup V}$ if and only if the pair (s, q) coincides with an appropriate pair (s_j, q_j), $j = 1, \ldots, k$ (Proposition 4). Therefore, according to (1), the V-restriction of the language accepted at state (s, z, q) of the automaton $A_{\uparrow V} \cap (P_{\text{perfect}})_{\uparrow W} \cap C_{\uparrow W\cup V}$ coincides with the language $L^z(P_{\text{perfect}})$. □

As an application example of Algorithm 1, we consider the context A and the specification C shown in Figs. 3(1) and 3(2), respectively. We assume that $V = \{u_1, u_2, v_1, v_2, v_3, x, y\}$ is the alphabet of a solution X to the equation $A \diamond_{\{i_1,i_2,o_1,o_2,x,y\}} X \cong C$. The solution B_1 to this equation, shown in Fig. 3(3), is not progressive. Therefore, we apply Step 1 of Algorithm 1 and we obtain the automaton Λ (A, B_1, C) shown in Fig. 4(2). Then, we apply Steps 2 and 3 and obtain the perfect automaton M_{perfect} shown in Fig. 5.

3.3.2. Deriving a largest progressive reduction

Given the trim deterministic context A and the trim deterministic specification C, let $P_{\text{p}} = \langle Z, V, \delta_{\text{p}}, z_0, F_{P_{\text{p}}}\rangle$ be a perfect automaton w.r.t. the automata A and C. In order to derive a largest progressive solution to the equation

$A \diamond_{Ext} X \cong C$ which is a reduction of P_p, we have to delete from the automaton P_p each state z that has a corresponding triplet (s, z, q) in the intersection $\Lambda(A, P_p, C)$ such that the external restriction of the language generated at (s, z, q) in not equal to the language generated at state q of C. However, after deleting all such states from P_p some other states of the intersection $\Lambda(A, P_p, C)$ might get the property that the external projection of the language generated at each of these states is not equal to the language generated at the corresponding state of the specification. Accordingly, we define, inductively, the notion of a non-progressive state.

A state (s, z, q) of the automaton $A_{\uparrow V} \cap P_{p\uparrow W} \cap C_{\uparrow W\cup V}$ is called 1-*non-progressive*, if the *Ext*-restriction of the language generated at the (s, z, q) does not coincide with the language generated by the specification C at state q, i.e., $L_{(s,z,q)}(A_{\uparrow V} \cap P_{p\uparrow W} \cap C_{\uparrow W\cup V})_{\downarrow Ext} \neq L_q(C)$. That is, there exists an external action $e \in Ext$ such that there is a transition from state q with the action e and there is no sequence $\beta \in (W \cup V)^*$ that is a prefix of a sequence in the language generated at state (s, z, q) such that $\beta_{\downarrow Ext} = e$. Suppose we have defined the *k-non-progressive* states for $k \geqslant 1$. A state (s, z, q) of the automaton $A_{\uparrow V} \cap P_{p\uparrow W} \cap C_{\uparrow W\cup V}$ is called $(k+1)$-*non-progressive* if (s, z, q) is *k-non-progressive* or if for some $e \in Ext$ such that there is a transition from state q with the action e, each state reached from (s, z, q) through a sequence $\beta \in (W \cup V)^*$ such that $\beta_{\downarrow Ext} = e$ is *k-non-progressive*. A state is *non-progressive* (or *not progressive*) if there exists a k such that the state (s, z, q) is k-non-progressive. Otherwise, a state is called *progressive*.

Given the equation $A \diamond_{Ext} X \cong C$, let P_p be a perfect automaton over alphabet V. A state z of P_p is non-progressive if there is a non-progressive triplet (s, z, q) in the automaton $\Lambda(A, P_p, C)$. We recall that each sequence that takes the automaton P_p to a non-progressive state cannot be included in a progressive solution. Thus, in order to get the largest progressive reduction of P_p, we are required to delete all non-progressive states from P_p. The largest submachine of P_p that does not have non-progressive states is the largest progressive solution to the equation. This submachine can be obtained from P_p by deleting its non-progressive states and all the states that become unreachable from the initial state. If the initial state of P_p is deleted, then none of the reductions of the automaton P_p is a progressive solution to the $A \diamond_{Ext} X \cong C$. Otherwise, the obtained submachine of P_p is the largest progressive solution to the equation $A \diamond_{Ext} X \cong C$.

Given a deterministic automaton P, we propose the following algorithm for deriving a largest progressive reduction of P.

Algorithm 2. Deriving a largest progressive reduction of P.

Input: A trim deterministic automaton $P = \langle R, V, \delta_P, r_0, F_P\rangle$, a trim deterministic context $A = \langle S, W, \delta_A, s_0, F_A\rangle$, and a trim deterministic specification $C = \langle Q, Ext, \delta_C, q_0, F_C\rangle$.

Output: A largest progressive reduction of P.

Step 1: Call Algorithm 1 that returns the perfect deterministic automaton P_{perfect} that is equivalent to P.

Step 2: Determine and delete all non-progressive states of the automaton P_{perfect} as described above. If the initial state of P_{perfect} is non-progressive (i.e., deleted), then the equation $A \diamond_{Ext} X \cong C$ has no progressive solutions that are reductions of P. Otherwise, the obtained submachine of P_{perfect} is the largest reduction of P that is a progressive solution of the equation.

As an example, consider the perfect automaton M_{perfect} for B_1 shown in Fig. 5, derived using Algorithm 1, and the corresponding automaton $\Lambda(A, M_{\text{perfect}}, C)$ shown in Fig. 6. States fG3, kH3, fK3, dM3 and kL3 of $\Lambda(A, M_{\text{perfect}}, C)$ are 1-non-progressive, thus we delete from M_{perfect} the states G, H, K, L, M, and we delete from $\Lambda(A, M_{\text{perfect}}, C)$ each state that has G, H, K, L, or M as a component. Afterwards, we observe that states cC2 and sE2 of $\Lambda(A, M_{\text{perfect}}, C)$ become non-progressive, i.e., these states are 2-non-progressive. Thus, we delete states C and E from M_{perfect} and obtain the automaton shown in Fig. 4(1). Moreover, we delete each state with a component C or E from $\Lambda(A, M_{\text{perfect}}, C)$. The obtained automaton $\Lambda(A, M_{\text{perfect}}, C)$ has only progressive states, thus the automaton of Fig. 4(1) is a largest reduction of B_1 that is a progressive solution.

Theorem 2. *Let P_p be a perfect automaton over alphabet V (w.r.t. the given context A and specification C). The largest submachine of P_p that does not contain non-progressive states (if such a submachine exists) is a largest reduction of P_p that is a progressive solution to the equation $A \diamond_{Ext} X \cong C$.*

As a corollary, if P_p is the largest submachine of a perfect largest solution to the equation $A \diamond_{Ext} X \cong C$ then a largest progressive reduction of P_p coincides with a largest progressive solution to the equation.

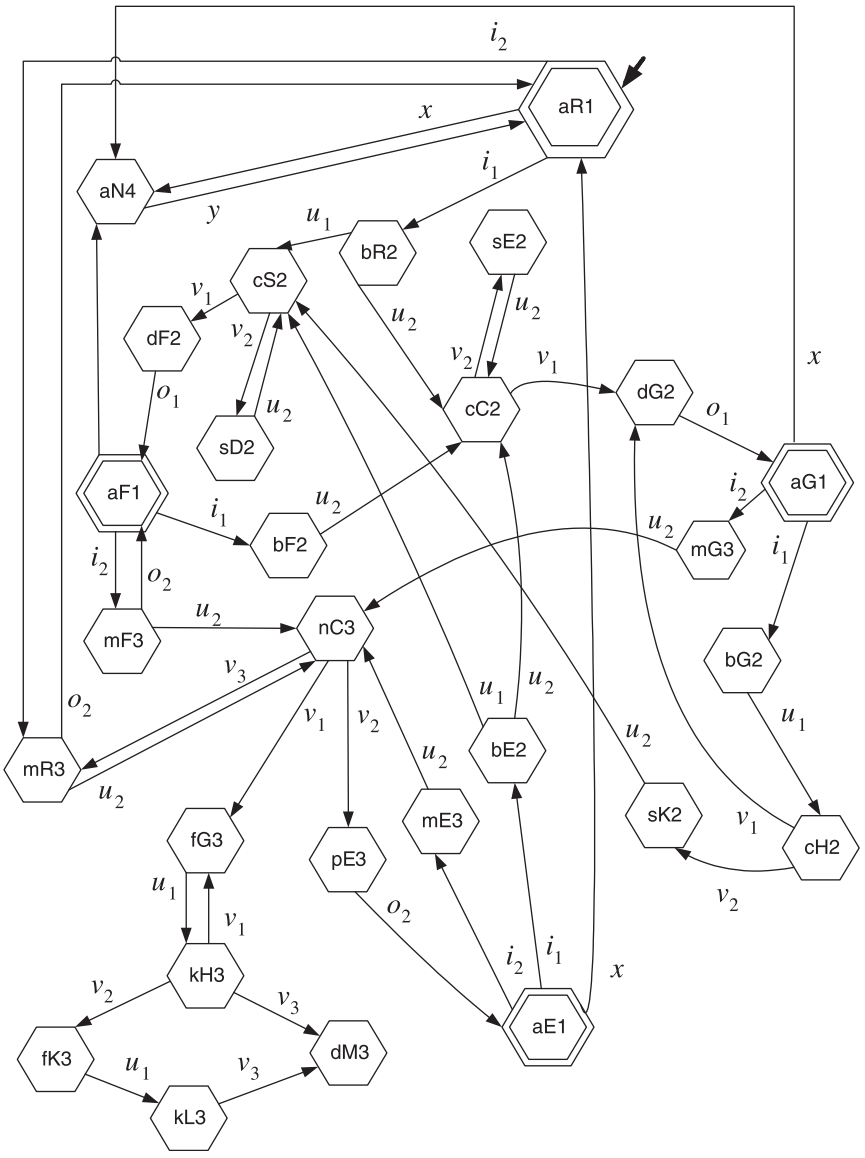


Fig. 6. Automaton $\Lambda\ (A, M_{\text{perfect}}, C)$.

Corollary 1. *The largest submachine of a perfect largest solution to the equation* $A \diamond_{Ext} X \cong C$ (*w.r.t. the automata A and C*) *that does not contain non-progressive states* (*if it exists*) *is a largest progressive solution to the equation.*

Proof of Theorem 2. Consider a non-progressive state z of P_{p} that induces an 1-non-progressive state (s, z, q) in $\Lambda\ (A, P_{\text{p}}, C)$. The V-restriction of each trace accepted by state (s, z, q) has to be deleted from the language of a largest progressive solution. A solution that has a trace accepted at the state z cannot be progressive. Since P_{p} is perfect this V-restriction coincides with the language accepted at state z of the automaton P_{p}, and thus, state z can be deleted from P_{p}. After deleting such non-progressive states from P_{p} we obtain a submachine P_{P1}. The automaton P_{P1} also is perfect, since a trace γ is deleted from P_{p} if and only if all the traces with the V-restriction γ are deleted from $\Lambda\ (A, P_{\text{p}}, C)$. If P_{P1} is not progressive the automaton $\Lambda\ (A, P_{\text{P1}}, C)$ has a 1-non-progressive state. Since the number of states of the automaton P_{p} is finite, the statement of the theorem can be easily proved by induction. □

We 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 accepting 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 $V \subset W \cup Ext$. Moreover, the procedure of deriving a largest progressive reduction of the automaton P over the alphabet $V = W \cup Ext$ is simpler than described above for the case when $V \subset W \cup Ext$. In this case, Step 2 of Algorithm 1 is skipped since the V-restriction of the automaton $\Lambda\,(A, P, C)$ coincides with the automaton $\Lambda\,(A, P, C)_{\downarrow V}$. Moreover, we do not have to compose the perfect automaton P_{p} of P with the specification and context in order to determine non-progressive states of P_{p} while deriving the largest progressive reduction of P_{p}. It is sufficient to delete non-progressive states directly from the automaton $\Lambda\,(A, P, C)$. The V-restriction of the obtained automaton (if it exists) is a largest progressive reduction of the automaton P over the alphabet $V = W \cup Ext$.

3.4. Characterization of all progressive solutions

A characterization of all progressive solutions to the equation $A \diamond_{Ext} X \cong C$ over I/O automata is proposed in [4]. In this section, we establish the conditions that allow us to describe all progressive solutions for a parallel automata equation. In other words, 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 [4]. We associate with each state of a largest progressive solution M_{L} a family of regular sets. In Section 3.2 we described how we obtain these sets. A reduction B of the largest progressive solution M_{L} is a progressive solution if and only if for each pair (t, r) of the intersection $B \cap M_{\mathrm{L}}$, the language of the reduction B at state t intersects each regular set of the family associated with the state r of M_{L}.

We recall that if a largest solution to a given automata equation is progressive then it is a largest progressive solution to the equation. Moreover, there exist an infinite number of equivalent largest solutions. Not each largest progressive solution can be used for the complete characterization of progressive solutions [3]. However, we show that a perfect largest progressive solution can be used for the complete characterization of all progressive solutions.

Given a deterministic largest progressive solution M_{L} with the state set R to the equation, 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 $B \cap M_{\mathrm{L}}$ of the automata M_{L} and B to the state pair (t, r). Since B is a reduction of the deterministic M_{L}, for a given t there always exists such an r (Proposition 1). Now, let (s, r, q) be a state of the automaton $\Lambda\,(A, M_{\mathrm{L}}, C)$ that is reachable from the initial state through the sequence β. Since M_{L} is progressive, each state (s, r, q) of the automaton $\Lambda\,(A, M_{\mathrm{L}}, C)$ 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 $Re[(s, r, q), e]_{\downarrow V}$ 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 $\Lambda\,(A, M_{\mathrm{L}}, C)$ (Proposition 5). Therefore, for (s, t, q) to be progressive, at least one sequence from the set $Re[(s, r, q), e]_{\downarrow V}$ has to be generated at state t.

Therefore, for each state (s, r, q) of the automaton $\Lambda(A, M_{\mathrm{L}}, C)$ 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}. For a reduction B of M_{L} to be progressive, it is sufficient that for each pair (t, r) in the intersection $B \cap M_{\mathrm{L}}$, 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 B satisfies the condition to be progressive.

Unfortunately, for an arbitrary largest solution M_{L}, we could have a progressive solution that does not satisfy this condition. This happens when there exists a triplet (s, r, q) of the automaton $\Lambda(A, M_{\mathrm{L}}, C)$ such that the V restriction of the language accepted at (s, r, q) is not equal to the language accepted at state r. Thus, the intersection $B \cap M_{\mathrm{L}}$ may have a pair (t, r) reachable by a sequence that is in the difference $L^r \backslash L_{\downarrow V}^{(s,r,q)}$. In this case, it may happen that for (s, r, q) of $\Lambda\,(A, M_{\mathrm{L}}, C)$ there does not exist a corresponding state (s, t, q) in $\Lambda\,(A, B, C)$. However, by construction, we have selected 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 state (s, t, q) in $\Lambda\,(A, B, C)$. The above situation cannot happen for a perfect largest solution M_{perfect} with the state set Z since the language accepted at state (s, z, q) of $\Lambda\,(A, M_{\mathrm{Perfect}}, C)$ equals the language accepted at state z of M_{perfect} (i.e.,

$L^r \backslash L^{(s,z,q)}_{\downarrow V} = \emptyset$). In other words, for every state (s, z, q) of $\Lambda(A, M_{\text{Perfect}}, C)$ there exists a corresponding state (s, t, q) in $\Lambda(A, B, C)$.

Proposition 8. *Given a trim deterministic context A and a trim deterministic specification C, let* $M_{\text{perfect}} = \langle Z, V, \delta_M, z_0, F_M \rangle$ *be a perfect automaton (w.r.t the automata A and C) that is equivalent to a largest progressive solution* M_{L} *to the equation* $A \diamond_{Ext} X \cong C$. *Let an automaton B be a trim reduction of* M_{perfect}, *a pair* (t, z) *be a state of the intersection* $B \cap M_{\text{perfect}}$, *and states s and q be states of the automata A and C, respectively. If the automaton* $\Lambda(A, M_{\text{perfect}}, C)$ *has a state* (s, z, q) *then the automaton* $\Lambda(A, B, C)$ *has a state* (s, t, q).

Proof. Let B be a reduction of M_{perfect} with the properties stated in the proposition, (t, z) be a state of the intersection $B \cap M_{\text{perfect}}$ and let (s, z, q) be a state of the automaton $\Lambda(A, M_{\text{perfect}}, C)$. Since M_{perfect} is a perfect automaton, $L^z(M_{\text{perfect}}) = L^{(s,z,q)}_{\downarrow V}$, i.e., for each sequence α of the language $L^z(M_{\text{perfect}})$ accepted at state z, there exists a sequence β with the V-restriction α that takes the automaton $\Lambda(A, M_{\text{perfect}}, C)$ to state (s, z, q).

Consider the sequence α of the language $L^z(M_{\text{perfect}})$ that takes the automaton B from the initial state to state t and a corresponding sequence β with the V-restriction α that takes the automaton $\Lambda(A, M_{\text{perfect}}, C)$ to state (s, z, q). Since the pair (t, z) is a state of the intersection $B \cap M_{\text{perfect}}$, such a sequence α exists. Due to Proposition 4, sequence β takes the automaton $\Lambda(A, B, C)$ from the initial state to state (s, t, q). □

Consequently, in order to have a complete characterization of all progressive solutions over a given alphabet, we use the perfect automaton M_{perfect} of a largest progressive solution M_{L}. By definition, if a triple (s, z, q) is not a state of the automaton $\Lambda(A, M_{\text{perfect}}, C)$, then for each external action $e \in Ext$ the set $Re[(s, z, q), e]$ is empty. Given an automaton M_{perfect}, we denote by $Re(z)$ the set of all non-empty sets $Re[(s, z, q), e]_{\downarrow V}$, for all $(s, q, e) \in S \times Q \times Ext$. An automaton $B = \langle T, V, \delta_B, t_0, F_B \rangle$ is progressive if for each pair of the intersection $M_{\text{perfect}} \cap B$, the prefix-closure of the language generated at state t of the automaton B intersects each set from $Re(z)$, i.e., the following holds:

$$Re(z) \neq \emptyset \Rightarrow \forall L \in Re(z)(L \cap L_t(B) \neq \emptyset).$$

The following theorem formally establishes the above necessary and sufficient conditions for a reduction of a largest progressive solution to be progressive.

Theorem 3. *Given a deterministic perfect largest progressive solution* M_{perfect} *to the equation* $A \diamond_{Ext} X \cong C$, *a trim automaton B is a progressive solution to the equation if and only if B is a reduction of the automaton* M_{perfect} *and for each state pair* (t, z) *in the intersection* $B \cap M_{\text{perfect}}$, *the prefix-closure of the language generated at state t of the automaton B intersects each set from* $Re(z)$.

Proof (*If part*). Let B be a reduction of M_{perfect} and for each state pair (t, z) in the intersection $B \cap M_{\text{perfect}}$, the prefix-closure of the language generated at state t of the automaton B intersects each set from $Re(z)$. Consider state (t, z) of the intersection $B \cap M_{\text{perfect}}$. Let state (s, t, q) be a state of the automaton $\Lambda(A, B, C)$. Then triplet (s, z, q) is a state of the automaton $\Lambda(A, M_{\text{perfect}}, C)$ (Proposition 5). Due to the stated conditions, for each action e defined at state q of the specification the set $Re[(s, z, q), e]_{\downarrow V}$ comprises the V-restriction of all prefixes of sequences generated at state (s, z, q) that have the Ext-restriction e. Since the prefix-closure of the language generated at state t intersects the set $Re[(s, z, q), e]$, at least one of these prefixes is generated at state (s, t, q). Therefore, the state (s, t, q) is progressive (Proposition 7).

(*Only if part*). Let B be a progressive solution. Each progressive solution is a reduction of a largest progressive solution, i.e., B is a reduction of M_{perfect}. Consider a state (t, z) of the intersection $B \cap M_{\text{perfect}}$. If triplet (s, z, q) is a state of the $\Lambda(A, M_{\text{perfect}}, C)$ then triplet (s, t, q) is a state of the $\Lambda(A, B, C)$ (Proposition 8). Moreover, the language generated at state (s, t, q) of the automaton $\Lambda(A, B, C)$ is a subset of that generated at state (s, z, q) of the automaton $\Lambda(A, M_{\text{perfect}}, C)$ (Proposition 5). Therefore, in order to have a state (s, t, q) progressive the prefix-closure of the language generated at state t has to intersect each set $Re[(s, z, q), e]_{\downarrow V}$ for each e that is defined at state q of the specification. □

4. Conclusions

In this paper we have addressed the problem of characterizing progressive solutions to an 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 that is possible in the specification. Given an alphabet of a solution and an automaton over this alphabet, we have proposed a technique for deriving a largest reduction of the automaton that is a progressive solution to the equation (if it exists). The technique can be used in order to determine a largest progressive solution. However, not each reduction of a largest progressive solution is progressive and therefore, the problem of characterizing all progressive solutions is not trivial. In this paper, we have established necessary and sufficient conditions that allow us to characterize all progressive solutions. The complete characterization of progressive solutions enables us to select an "optimal" solution according to different criteria. For example, an optimal solution may be defined as the one with the smallest number of states, actions or transitions, or as the fastest one, that is, the solution that, when combined with the context, executes the external actions with a shortest internal dialog between the components.

Acknowledgments

The authors would like to thank the anonymous reviewers and editor Dr. Zoltán Ésik for their helpful comments in improving this manuscript.

References

[1] G. Barrett, S. Lafortune, Bisimulation, the supervisory control problem, and strong model matching for finite state machines, Discrete Event Dynamic Systems: Theory Appl. 8 (4) (1998) 377–429.
[2] G.v. Bochmann, P.M. Merlin, On the construction of communication protocols, ICCC, 1980, pp. 371–378, reprinted, in: C. Sunshine (Ed.), Communication Protocol Modeling, Artech House, Norwood, MA, 1981.
[3] S. Buffalov, K. El-Fakih, N. Yevtushenko, G.v. Bochmann, Progressive solutions for a parallel automata equation, Proc. IFIP 23rd Internat. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE '03), Lecture Notes in Computer Science, Vol. 2767, Springer, Berlin, 2003, pp. 367–382.
[4] J. Drissi, G.v. Bochmann, Submodule construction for systems of I/O automata, Technical Report #1133, DIRO, Université de Montréal, Canada, 1999.
[5] K. El-Fakih, N. Yevtushenko, Fault propagation by equation solving, Proc. IFIP 24th Internat. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE '04), Lecture Notes in Computer Science, Vol. 3235, Springer, Berlin, 2004, pp. 185–198.
[6] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[7] S.G.H. Kelekar, Synthesis of protocols and protocol converters using the submodule construction approach, in: A. Danthine et al. (Eds.), Protocol Specification, Testing, and Verification—PSTV XIII, 1994.
[8] R. Kumar, S. Nelvagal, S.I. Marcus, A discrete event systems approach for protocol conversion, Discrete Event Dynamical Systems: Theory Appl. 7 (3) (1997) 295–315.
[9] P. Merlin, G.v. Bochman, On the construction of submodule specifications and communication protocols, ACM Trans. Programming Lang. Systems 5 (1) (1983) 1–25.
[10] J. Parrow, Submodule construction as equation solving in CCS, Theoret. Comput. Sci. 68 (1989).
[11] A. Petrenko, N. Yevtushenko, Solving asynchronous equations, in: S. Bukowski, A. Cavalli, E. Najm (Eds.), Formal Description Techniques and Protocol Specification, Testing, and Verification—FORTE XI/PSTVXVIII '98, Chapman & Hall, London, 1998, pp. 231–247.
[12] A. Petrenko, N. Yevtushenko, G.v. Bochmann, R. Dssouli, Testing in context: framework and test derivation, Comput. Comm. J. Special issue on Protocol Eng. 19 (1996) 1236–1249.
[13] H. Qin, P. Lewis, Factorisation of finite state machines under strong and observational equivalences, J. Formal Aspects Comput. 3 (1991) 284–307.
[14] Z. Tao, G.v. Bochmann, R. Dssouli, A formal method for synthesizing optimized protocol converters and its application to mobile data networks, Mobile Networks Appl. 2 (3) (1997) 259–269.
[15] W.M. Wonham, P.J. Ramadge, On the supremal controllable sublanguage of a given language, SIAM J. Control Optim. 25 (3) (1987) 637–659.
[16] N. Yevtushenko, T. Villa, R.K. Brayton, A. Petrenko, A. Sangiovanni-Vincentelli, Solution of parallel language equations for logic synthesis, in: Proc. Internat. Conf. on Computer-Aided Design, 2001, pp. 103–110.
[17] N. Yevtushenko, T. Villa, A. Petrenko, R.K. Brayton, A. Sangiovanni-Vincentelli, Logic synthesis by equation solving, Technical Report, Tomsk State University, Russia, 1999, p. 27 (in Russian).