Compositionally Progressive Solutions of Synchronous Language Equations

Nina Yevtushenko[¶] Tiziano Villa[§,†] Robert K. Brayton[‡] Alex Petrenko[¶¶]
Alberto L. Sangiovanni-Vincentelli[†,‡]

[¶]Dept. of EECS
Tomsk State University
Tomsk, 634050, Russia

[§]DIEGM
Univ. of Udine
33100 Udine, Italy

[†]PARADES
Via di S.Pantaleo, 66
00186 Roma, Italy

[‡]Dept. of EECS
Univ. of California
Berkeley, CA 94720

[¶¶]CRIM
550 Sherbrooke West
Montreal, H3A 1B9, Can

April 29, 2003

Abstract

The paper addresses the problem of designing a component that combined with a known part of a system, called the context FSM, is a reduction of a given specification FSM. We study compositionally progressive solutions of synchronous FSM equations. Such solutions, when combined with the context, do not block any input that may occur in the specification, so they are of practical use. We show that if a synchronous FSM equation has a compositionally progressive solution, then the equation has the largest compositionally progressive solution. However, not each reduction of the largest compositionally progressive solution is compositionally progressive. Since generally the number of reductions is infinite, the problem of characterizing all progressive solutions is non-trivial.

1 Introduction

The paper addresses the problem of designing a component that combined with a known part of a system, called the context FSM, is a reduction of a given specification FSM. In [2, 3] we proposed a general frame based on solving equations over languages associated to the automata modelling a given system. We introduced two composition operators for abstract languages: synchronous composition, $\bullet$, and parallel composition, $\diamond$, and we studied the most general solutions of the language equations $A \bullet X \subseteq C$ and $A \diamond X \subseteq C$ ($\subseteq$ denotes language containment).

Here we study compositionally progressive solutions of synchronous FSM equations. Such solutions, when combined with the context, do not block any input that may occur in the specification, so they are of practical use. We show that if a synchronous FSM equation has a compositionally progressive solution, then the equation has the largest compositionally progressive solution. However, not each reduction of the largest compositionally progressive solution is compositionally progressive. Since generally the number of reductions is infinite, the problem of characterizing all progressive solutions is non-trivial.

Basic definitions about equations over languages are provided in Sec. 2. An algorithm that deletes compositionally non-progressive sequences is given in Sec. 3. A different algorithm based on state splitting is presented in Sec. 4.

2 Equations over Languages

2.1 Languages

We remind the notions of substitution and homomorphism of languages, referring to a standard textbook for the basic definitions [1]. A **substitution** f is a mapping of an alphabet Σ onto subsets of $\Delta^\star$ for some alphabet Δ. The substitution f is extended to strings by setting $f(\epsilon) = \epsilon$ and $f(xa) = f(x)f(a)$. An **homomorphism** h is a substitution such that $h(a)$ is a single string for each symbol a in the alphabet Σ. Useful operations on languages are:

1. Given a language L over alphabet $X \times V$, consider the homomorphism $p : X \times V \to V$ defined as $p((x, v)) = v$, then the language $L_{\downarrow V} = \{p(\alpha) \mid \alpha \in L\}$ over alphabet V is the **projection** of language L to alphabet V, or V-projection of L. By definition of substitution $p(\epsilon) = \epsilon$.

2. Given a language L over alphabet X and an alphabet V, consider the substitution $l : X \to X \times V$ defined as $l(x) = \{(x, v) \mid v \in V\}$, then the language $L_{\uparrow V} = \{l(\alpha) \mid \alpha \in L\}$ over alphabet $X \times V$ is the **lifting** of language L to alphabet V, or V-lifting of L. By definition of substitution $l(\epsilon) = \epsilon$.

Proposition 2.1 *(a) Let L_1, L_2 be languages over alphabet U. Then $\uparrow$ commutes with $\cap$*

$$(L_1 \cap L_2)_{\uparrow I} = L_{1\uparrow I} \cap L_{2\uparrow I}.$$

(b) Let M_1, M_2 be languages over alphabet $I \times U$. If $M_2 = (M_{2\downarrow U})_{\uparrow I}$ (or $M_1 = (M_{1\downarrow U})_{\uparrow I}$) then $\downarrow$ commutes with $\cap$

$$(M_1 \cap M_2)_{\downarrow U} = M_{1\downarrow U} \cap M_{2\downarrow U}.$$

Proof. The proof is available in an extended version. □

Definition 2.1 *A language L over alphabet X is* **prefix-closed** *if* $\forall \alpha \in X^\star \; \forall x \in X \; [\alpha x \in L \Rightarrow \alpha \in L]$.

Definition 2.2 *A language L over alphabet $X = I \times O$ is I-*__progressive__ *if*

$$\forall \alpha \in X^\star \; \forall i \in I \; \exists o \in O \; [\alpha \in L \Rightarrow \alpha \, (i, o) \in L].$$

2.2 Synchronous Composition of Languages

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

Definition 2.3 *Given alphabets I, U, O, language L_1 over $I \times U$ and language L_2 over $U \times O$, the* **synchronous composition** *of languages L_1 and L_2 is the language* [1] *$[(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}]_{\downarrow I \times O}$, denoted by $L_1 \bullet L_2$, defined over $I \times O$.*

Definition 2.4 *Given a language A over alphabet $I \times U$, a language B over alphabet $U \times O$ is A-*__compositionally__ *I-*__progressive__ *if the language $L = A_{\uparrow O} \cap B_{\uparrow I}$ over alphabet $X = I \times U \times O$ is I-progressive, i.e., $\forall \alpha \in X^\star \; \forall i \in I \; \exists (u, o) \in U \times O \; [\alpha \in L \Rightarrow \alpha \, (i, u, o) \in L]$.*

2.3 Solution of Language Equations under Synchronous Composition

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

$$X \bullet A \subseteq C. \tag{1}$$

[1] Use the same order $I \times U \times O$ in the languages $(L_1)_{\uparrow O}$ and $(L_2)_{\uparrow I}$.

Definition 2.5 *Given alphabets I, U, O, a language A over alphabet $I \times U$ and a language C over alphabet $I \times O$, language B over alphabet $U \times O$ is called a* **solution** *of the equation $X \bullet A \subseteq C$ iff $B \bullet A \subseteq C$. The* **largest solution** *is the solution that contains any other solution. $B = \emptyset$ is the trivial solution.*

Theorem 2.1 *The largest solution of the equation $A \bullet X \subseteq C$ is the language $S = \overline{A \bullet \overline{C}}$. A language B over alphabet $U \times O$ is a solution of $X \bullet A \subseteq C$ iff $B \subseteq \overline{A \bullet \overline{C}}$.*

2.4 Solution of FSM Language Equations under Synchronous Composition

Definition 2.6 *A* **finite state machine** *(FSM) is a 5-tuple $M = \langle S, I, O, T, r \rangle$ where S represents the finite state space, I represents the finite input space, O represents the finite output space and $T \subseteq I \times S \times S \times O$ is the transition relation. On input i, the FSM at present state p may transit to next state n and produce output o iff $(i, p, n, o) \in T$. State $r \in S$ represents the initial or reset state. We may use δ instead of T_n and λ instead of T_o. If at least one transition is specified for each present state and input pair, the FSM is said to be* **complete***. If no transition is specified for at least one present state and input pair, the FSM is said to be* **partial***. An FSM is said to be* **trivial** *when $T = \emptyset$, denoted by M_ϵ.*

Definition 2.7 *An FSM $M' = \langle S', I', O', T', r' \rangle$ is a submachine of FSM $M = \langle S, I, O, T, r \rangle$ if $S \subseteq S$, $I' \subseteq I$, $O' \subseteq O$, $r' = r$, and $T' \subseteq T$, i.e., T' is a restriction of T to the domain of definition $I' \times S' \times S' \times O'$.*

Definition 2.8 *Given an FSM $M = \langle S, I, O, T, r \rangle$, consider the finite automaton $F(M) = \langle S, I \times O, \Delta, r, S \rangle$, where $((i, o), s, s') \in \Delta$ iff $(i, s, s', o) \in T$. The language accepted by $F(M)$ is denoted $L_r^\times(M)$, and by definition is the* **$\times$-language** *of M at state r. Similarly $L_s^\times(M)$ denotes the language accepted by $F(M)$ when started at state s, and by definition is the $\times$-language of M at state s.*

$\epsilon \in L_r(M)$ because the initial state is accepting. An FSM M is **trivial** iff $L_r(M) = \{\epsilon\}$.

Definition 2.9 *A language L is an* **FSM language** *if there is an FSM M such that the associated automaton $F(M)$ accepts L. The language associated to a DFSM is sometimes called a* **behaviour** [2].

Definition 2.10 *State t of FSM M_B is said to be a* **reduction** *of state s of FSM M_A (M_A and M_B are assumed to have the same input/output set), written $t \preceq s$, iff $L_t(M_B) \subseteq L_s(M_A)$. States t and s are* **equivalent states***, written $t \cong s$, iff $t \preceq s$ and $s \preceq t$, i.e., when $L_t(M_B) = L_s(M_A)$. An FSM with no two equivalent states is a* **reduced** *FSM.*

Similarly, M_B is a **reduction** *of M_A, $M_B \preceq M_A$, iff r_{M_B}, the initial state of M_B, is a reduction of r_{M_A}, the initial state of M_A. When $M_B \preceq M_A$ and $M_A \preceq M_B$ then M_A and M_B are* **equivalent machines***, i.e., $M_A \cong M_B$.*

In the sequel we will need the following procedure.

Procedure 2.1 *Input: FSM Language L^{FSM} over $I \times O$; Output: Largest I-progressive FSM language $Prog(L^{FSM})$ over $I \times O$.*

1. Build a deterministic automaton A accepting L^{FSM}.
2. Iteratively delete all states that have an undefined transition for some input (meaning: states such that $\exists i \in I$ with no $o \in O$ for which there is an outgoing edge carrying the label (i, o)), together with their incoming edges, until the initial state is deleted or no more state can be deleted.
3. If the initial state has been deleted, then $Prog(L^{FSM}) = \emptyset$. Otherwise, let $\hat{A}$ be the automaton produced by the procedure and $Prog(L^{FSM})$ the language that $\hat{A}$ accepts. Any I-progressive FSM language in L^{FSM} must be a subset of $Prog(L^{FSM})$.

[2] The language associated to a NDFSM includes a set of behaviours.

3 Largest FSM Compositional Solutions

Let $I = I_1 \times I_2$ and $O = O_1 \times O_2$. It is interesting to compute the subset of compositionally I-progressive solutions B, i.e., such that $A_{\uparrow I_2 \times O_2} \cap B_{\uparrow I_1 \times O_1}$ is an I-progressive FSM language $\subseteq (I \times U \times V \times O)^\star$. Thus the composition (after projection on the external signals) is the language of a complete FSM over inputs $I_1 \times I_2$ and outputs $O_1 \times O_2$. Since $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{\uparrow I_1 \times O_1}$ is prefix-closed and hence corresponds to a partial FSM, we have to restrict it so that it is also I-progressive, which corresponds to a complete FSM. If S^{FSM} is compositionally I-progressive, then S^{FSM} is the largest compositionally I-progressive solution of the equation. However, not every non-empty subset of S^{FSM} inherits the feature of being compositionally I-progressive. If S^{FSM} is not compositionally I-progressive, then denote by $cProg(S^{FSM})$ the largest compositionally I-progressive subset of S^{FSM}. Conceptually $cProg(S^{FSM})$ is obtained from S^{FSM} by deleting each string α such that, for some $i \in I$, there is no $(u, v, o) \in U \times V \times O$ for which it holds $\alpha\,(i, u, v, o) \in A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{\uparrow I_1 \times O_1}$. The following procedure tells how to compute $cProg(S^{FSM})$.

Procedure 3.1 *Input: Largest prefix-closed solution S^{FSM} of synchronous equation $A \bullet X \subseteq C$ and context A; Output: Largest compositionally I-progressive prefix-closed solution $cProg(S^{FSM})$.*

1. Initialize i to 1 and S^i to S^{FSM}.
2. Compute $R^i = A_{\uparrow I_2 \times O_2} \cap S^i_{\uparrow I_1 \times O_1}$.
   If the language R^i is I-progressive then $cProg(S^{FSM}) = S^i$.
   Otherwise
   (a) Obtain $Prog(R^i)$, the largest I-progressive subset of R^i. by using Proc. 2.1.
   (b) Compute $T^i = S^i \setminus (R^i \setminus Prog(R^i))_{\downarrow I_2 \times U \times V \times O_2}$.
3. If $T^{i\ FSM} = \emptyset$ then $cProg(S^{FSM}) = \emptyset$.
   Otherwise
   (a) Assign the language $T^{i\ FSM}$ to S^{i+1}.
   (b) Increment i by 1 and go to 2.

Theorem 3.1 *Proc. 3.1 returns the largest compositionally I-progressive (prefix-closed) solution, if it terminates.*

Proof. The proof is available in an extended version. □

Example 3.1 *Given the FSMs M_A and M_C in Fig. 1(a)-(b), the largest language solution S in Fig. 1(c) of the equation $M_A \bullet M_X \preceq M_X$ is U-progressive, but not compositionally I-progressive, because $R^1 = A \cap S^1_{\uparrow I_1 \times O_1}$ in Fig. 1(d) is not I-progressive ($S^1 = S^{FSM}$), given that there is no transition under input i_2 from the state labeled by is. The steps of Proc. 3.1, Figs. 1(e)-(f) and 2(a)-(b), show the computations to find S^2; since $R^2 = A \cap S^2_{\uparrow I_1 \times O_1}$ in Fig. 2(c) is I-progressive, S^2 is compositionally I-progressive and $cProg(S^{FSM}) = S^2$. Finally M_{S^2} in Fig. 2(d) is the largest compositionally I-progressive FSM solution.*

We will prove next that Proc. 3.1 terminates. To provide some intuition we analyze the structure of the automaton recognizing R^2, as it is derived from the one recognizing R^1. Consider the languages R^1 and R^2, where $R^1 = A_{\uparrow I_2 \times O_2} \cap S^1_{\uparrow I_1 \times O_1}$, and $R^2 = A_{\uparrow I_2 \times O_2} \cap S^2_{\uparrow I_1 \times O_1}$. The language R^1 is recognized by the DFA $F^1 = \langle S^1, I_1 \times I_2 \times U \times V \times O_1 \times O_2, \Delta^1, r^1, Q^1 \rangle$, and each accepting state $q^1_j \in Q^1,\ j \in J^1 = \{1, \ldots, |Q^1|\}$, recognizes the language $L(q^1_j)$ [3]. Note that $Q^1 = Q^1_P \cup Q^1_{NP}$, where Q^1_P is the subset of progressive states and Q^1_{NP} is the subset of non-progressive states. The following derivation expresses R^2 as a function of R^1:

$$\begin{aligned} R^2 &= A_{\uparrow I_2 \times O_2} \cap S^2_{\uparrow I_1 \times O_1} \\ &= A_{\uparrow I_2 \times O_2} \cap [S^1 \setminus (R^1 \setminus Prog(R^1))_{\downarrow I_2 \times U \times V \times O_2}]_{\uparrow I_1 \times O_1} \end{aligned}$$

[3] We recall that given an FA $F = \langle S, \Sigma, \Delta, r, Q \rangle$, the language accepted or recognized by $s \in S$, denoted $L_r(F|s)$ or $L_r(s)$, is the set of words whose runs from r reach s.

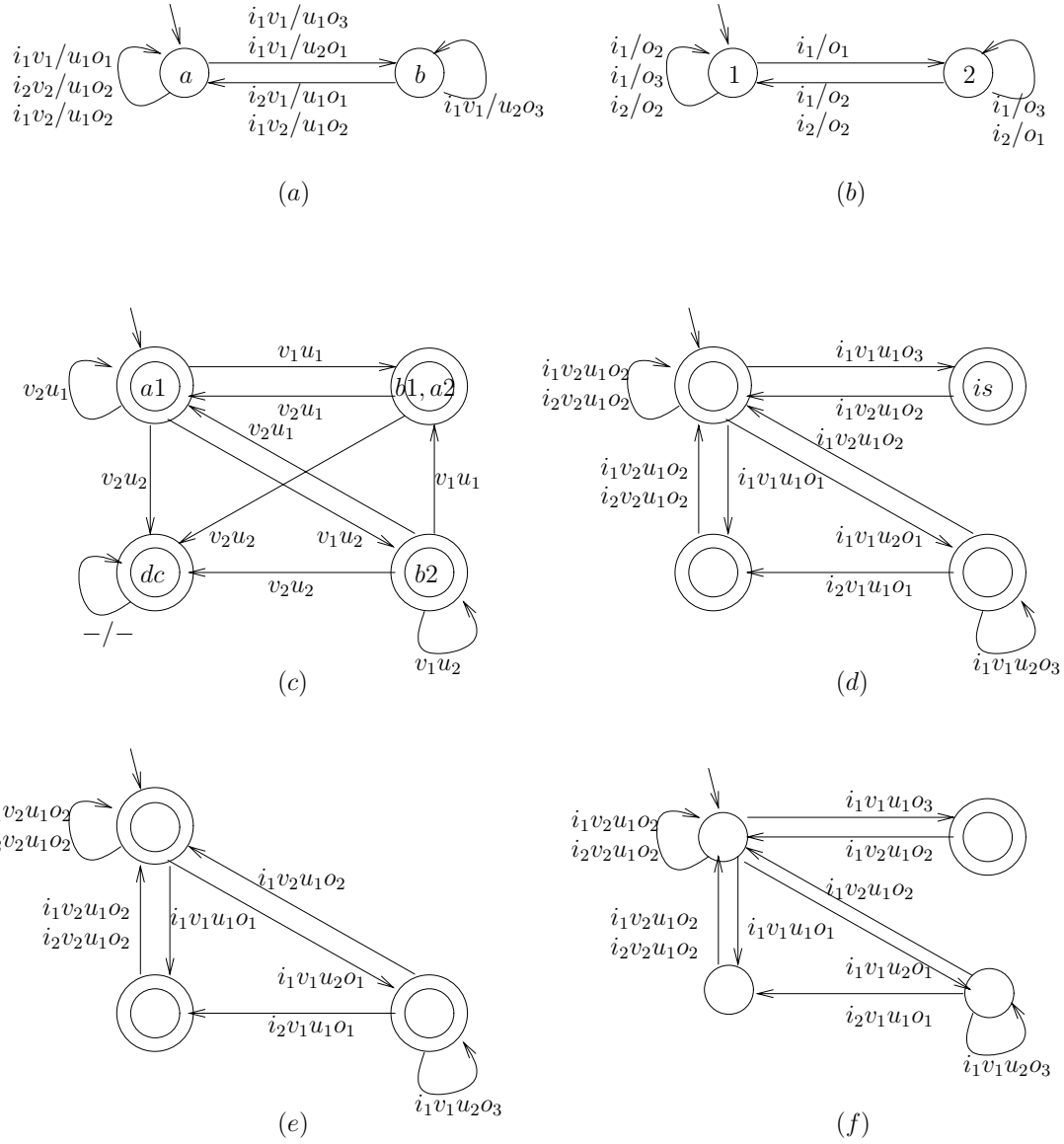

Figure 1: Illustration of Example 3.1. (a) FSM M_A; (b) FSM M_C; (c) Largest language solution $S = \overline{A \bullet \overline{C}}$; (d) $R^1 = A \cap S^1_{\uparrow I_1 \times O_1}$, with $S^1 = S^{FSM}$; (e) $Prog(R^1)$; (f) $R^1 \setminus Prog(R^1)$.

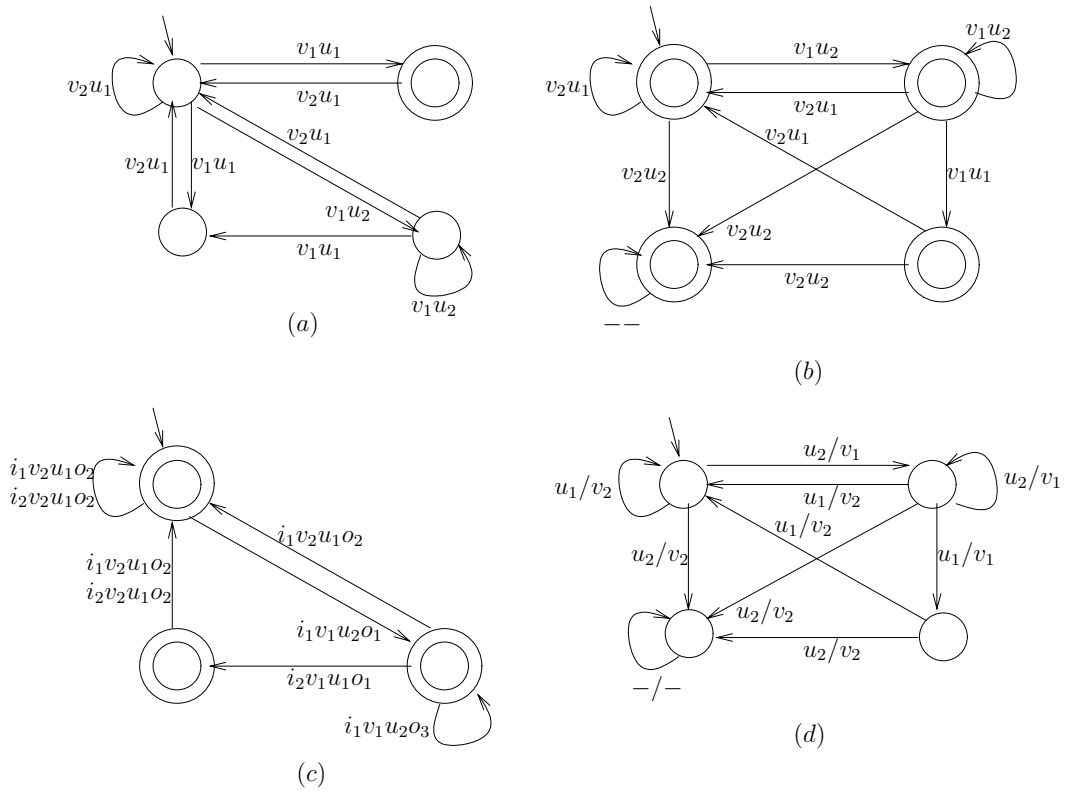

Figure 2: Illustration of Example 3.1. (a) $B^1 = (R^1 \setminus Prog(R^1))_{\downarrow V\times U}$; (b) $T^1 = S^1 \setminus B^1$; (c) $R^2 = A \cap S^2_{\uparrow I_1\times O_1}$, with $S^2 = T^{1\ FSM}$; (d) Largest compositionally progressive FSM solution M_{S^2}.

$$
\begin{aligned}
&= A_{\uparrow I_2\times O_2} \cap [S^1 \cap \overline{(R^1 \setminus Prog(R^1))_{\downarrow I_2\times U\times V\times O_2}}]_{\uparrow I_1\times O_1} \\
&= A_{\uparrow I_2\times O_2} \cap S^1_{\uparrow I_1\times O_1} \cap (\overline{(R^1 \setminus Prog(R^1))_{\downarrow I_2\times U\times V\times O_2}})_{\uparrow I_1\times O_1} \\
&= A_{\uparrow I_2\times O_2} \cap S^1_{\uparrow I_1\times O_1} \cap \overline{B}^1_{\uparrow I_1\times O_1} \\
&= R^1 \cap \overline{B}^1_{\uparrow I_1\times O_1}
\end{aligned}
$$

using the fact that $(L_1 \cap L_2)_{\uparrow I} = L_{1\uparrow I} \cap L_{2\uparrow I}$ by Prop. 2.1.

Let us examine the constructive steps leading from language R^1 to language R^2.

- Language $R^1 \setminus Prog(R^1)$. It is recognized by the DFA $F^1_{NP} = \langle S^1, I_1 \times I_2 \times U \times V \times O_1 \times O_2, \Delta^1, r^1, Q^1_{NP}\rangle$, where Q^1_{NP} is the subset of non-progressive states of Q^1. Each accepting state $q^1 \in Q^1_{NP}$ recognizes the language $L(q^1)$.

- Language $B^1 = (R^1 \setminus Prog(R^1))_{\downarrow I_2\times U\times V\times O_2}$. It is recognized by the NDFA $F^1_{B_1} = \langle S^1, I_2 \times U \times V \times O_1, \Delta^1, r^1, Q^1_{NP}\rangle$, that is the projection of F^1_{NP} onto $I_2 \times U \times V \times O_2$. Each accepting state $q^1 \in Q^1_{NP}$ recognizes the language $L(q^1)_{\downarrow I_2\times U\times V\times O_2}$.

- Language $R^2 = R^1 \cap (\overline{((R^1 \setminus Prog(R^1))_{\downarrow I_2\times U\times V\times O_2}})_{\uparrow I_1\times O_1} = R^1 \cap \overline{B}^1_{\uparrow I_1\times O_1}$. To complement B^1, the NDFA of B^1 must be determinized yielding a DFA $F^1_{\hat{B}^1} = \langle \hat{S}^1, I_2 \times U \times V \times O_1, \hat{\Delta}^1, \hat{r}^1, \hat{Q}^1\rangle$ whose states are subsets of states of B^1; each state of the determinized automaton recognizes a language that is the intersection of the languages recognized by the states in the related subset. Therefore each accepting state $\hat{q}^1_j \in \hat{Q}^1,\ j \in \hat{J}^1 =$

$\{1,\ldots,|\hat{Q}^1|\}$ is such that $\hat{q}^1_j = \bigcup\{q^1_k \mid k \in K^1_j \subseteq J\}$ and $L(\hat{q}^1_j) = \bigcap_{k\in K^1_j\subseteq J^1} L(q^1_k)_{\downarrow I_2\times U\times V\times O_2}$. Moreover, a *don't care* state $\{dc\}$ is added, that recognizes the language obtained as the complement of the union of the languages recognized by the states of the determinized automaton:

$$L(\{dc\}) = \overline{\bigcup_{j\in\hat{J}^1} L(\hat{q}^1_j)} = \bigcap_{j\in\hat{J}^1} \overline{L(\hat{q}^1_j)} = \bigcap_{j\in\hat{J}^1} \overline{\bigcap_{k\in K^1_j\subseteq J^1} L(q^1_k)_{\downarrow I_2\times U\times V\times O_2}}.$$

$$= \bigcap_{j\in\hat{J}^1} \bigcup_{k\in K^1_j\subseteq J^1} \overline{L(q^1_k)_{\downarrow I_2\times U\times V\times O_2}} = \bigcup_{H\in\mathcal{H}\subseteq 2^{J^1}} \bigcap_{h\in H\subseteq J^1} \overline{L(q^1_h)_{\downarrow I_2\times U\times V\times O_2}}.$$

The DFA of $\overline{B}^1$ is obtained by switching accepting and non-accepting states of the determinization of B^1. Finally the language R^2 is recognized by the DFA $F^2 = \langle S^2, I_1\times I_2\times U\times V\times O_1\times O_2, \Delta^2, r^2, Q^2\rangle$, obtained by intersection of the automata of R^1 and $\overline{B}^1_{\uparrow I_1\times O_1}$. So each accepting state $q^2_j \in Q^2,\ j \in J^2 = \{1,\ldots,|Q^2|\}$, recognizes the language $L(q^2_j)$ expressed in the following form:

$$\bigcup_{H,K\in\mathcal{H}^2_j\subseteq 2^{J^1}} [\, L(q^1_{l^2_j}) \bigcap_{h\in H\subseteq J^1} (L(q^1_h)_{\downarrow I_2\times U\times V\times O_2})_{\uparrow I_1\times O_1} \bigcap_{k\in K\subseteq J^1} (\overline{L(q^1_k)_{\downarrow I_2\times U\times V\times O_2}})_{\uparrow I_1\times O_1}\,],$$

where $l^2_j \in J^1, \mathcal{H}^2_j \subseteq 2^{J^1}$, and $q^1_{l^2_j}, q^1_h \in Q^1$. We used the fact that $(L_1\cup L_2)_{\uparrow I} = L_{1\uparrow I}\cup L_{2\uparrow I}$ and $(L_1\cap L_2)_{\uparrow I} = L_{1\uparrow I}\cap L_{2\uparrow I}$ by Prop. 2.1.

The bottom line is the observation that the states of the DFA accepting R^2 recognize languages that are combinations through a number of basic operators of the languages recognized by the states of the DFA accepting R^1. This fact can be established also for the successive iterations R^i, meaning that also the languages of the states of the DFA accepting R^i can be expressed as a finite combination of the languages of the states of the DFA accepting R^1; this property can deliver a proof of termination, when cast in the frame of an induction argument.

Theorem 3.2 *Proc. 3.1 terminates.*

Proof. The proof is available in an extended version. □

4 An Alternative Algorithm

We are currently working on a new algorithm based on state splitting. We provide here the motivation and a sketch of it, referring to an extended version for the complete results.

An input-output sequence $\alpha\beta$ of the largest solution is called *compositionally non-progressive* if every solution accepting the sequence is not compositionally progressive. In other words, such a sequence *induces* a compositionally non-progressive solution. An input-output sequence $\alpha\beta$ is called *compositionally progressive* is there exists a compositionally progressive solution accepting the sequence. This suggests two options: either we delete all compositionally non-progressive sequences from the largest solution or we "split" states of the largest solution into collections of equivalent states such that each state of the refined automaton S_{split} is reachable from the initial state either only by compositionally progressive or only by compositionally non-progressive sequences. A sub-automaton of S_{split} that includes all compositionally progressive sequences is the largest compositionally progressive solution, if it exists. Therefore there are two approaches to find the largest compositionally progressive solution and we are investigating both.

The one based on deleting non-progressive sequences from the largest solution has been presented in the previous section and required a difficult proof of termination (potentially there are infinite sequences to be deleted). We are going to sketch next the approach based on state splitting. The automaton with split states could be further used for a complete characterization of the compositionally progressive solutions of a given equation.

The following procedure tells how to compute S^{FSM}_{split}.

Procedure 4.1 *Input: Largest prefix-closed solution S^{FSM} of synchronous equation $A \bullet X \subseteq C$ and context A; Output: S^{FSM}_{split}, i.e., largest prefix-closed solution such that each state of S^{FSM}_{split} is reachable from the initial state either only by compositionally progressive or only by compositionally non-progressive sequences.*

1. Initialize S to S^{FSM}.
2. Compute $R = A_{\uparrow I_2 \times O_2} \cap S_{\uparrow I_1 \times O_1}$.
3. S_{split} is the automaton obtained by determinizing $R_{\downarrow I_2 \times U \times V \times O_2}$.
4. Add to S_{split} the dc state (self-looping under all inputs), and the following transitions to it: for each transition in S under $i_2 u v o_2$ from state q to state dc, add a transition in S_{split} under $i_2 u v o_2$ from each state containing a state $(s_i, q), s_i \in A$, to state dc. Set S^{FSM}_{split} to S_{split}.

Theorem 4.1 *The automaton S^{FSM}_{split} returned by Proc. 4.1 is such that*

1. *S^{FSM}_{split} is equivalent to S^{FSM}.*
2. *For each state r of S^{FSM}_{split}, if there is a compositionally non-progressive sequence that takes S^{FSM}_{split} from the initial state to r, then each sequence that takes S^{FSM}_{split} from the initial state to r is compositionally non-progressive.*

The following procedure tells how to compute $cProg(S^{FSM})$.

Procedure 4.2 *Input: Largest prefix-closed solution S^{FSM}_{split} of synchronous equation $A \bullet X \subseteq C$ and context A; Output: Largest compositionally I-progressive prefix-closed solution $cProg(S^{FSM})$.*

1. Initialize i to 1 and S^i to S^{FSM}_{split}.
2. Compute $R^i = A_{\uparrow I_2 \times O_2} \cap S^i_{\uparrow I_1 \times O_1}$.
3. If the language R^i is I-progressive then $cProg(S^{FSM}) = S^i$.
   Otherwise
   (a) Determine the set of incompletely specified states (with respect to I) $B = \{(s, r)\}$, $(s, r) \in R^i, s \in A_{\uparrow I_2 \times O_2}, r \in S^i_{\uparrow I_1 \times O_1}$.
   (b) Obtain S^{i+1}, by deleting from S^i each state r such that $\exists s, s \in A_{\uparrow I_2 \times O_2}$, for which $(s, r) \in B$.
   (c) If the initial state is deleted then $cProg(S^{FSM}) = \emptyset$.
   Otherwise
      i. Obtain R^{i+1}, by deleting from R^i each state $(s', r), s' \in A_{\uparrow I_2 \times O_2}, r \in S^i_{\uparrow I_1 \times O_1}$, such that $\exists s, s \in A_{\uparrow I_2 \times O_2}$, for which $(s, r) \in B$.
      ii. Increment i by 1 and go to 3.

Theorem 4.2 *Proc. 4.2 returns the largest compositionally I-progressive (prefix-closed) solution.*

References

[1] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[2] N. Yevtushenko, T. Villa, R. Brayton, A. Petrenko, and A. Sangiovanni-Vincentelli. Solution of parallel language equations for logic synthesis. In *The Proceedings of the International Conference on Computer-Aided Design*, pages 103–110, November 2001.

[3] N. Yevtushenko, T. Villa, R. Brayton, A. Petrenko, and A. Sangiovanni-Vincentelli. Solution of synchronous language equations for logic synthesis. In *The Biannual 4th Russian Conference with Foreign Participation on Computer-Aided Technologies in Applied Mathematics*, September 2002.