# Compositionally Progressive Solutions of Synchronous FSM Equations

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

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

[§]DI, Un. di Verona
Strada Le Grazie, 15
37134 Verona, 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

June 8, 2007

**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 a largest compositionally progressive solution. We provide two different algorithms to compute a largest compositionally progressive solution: one deletes all compositionally non-progressive strings from a largest solution, the other splits states of a largest solution and then removes those inducing a non-progressive composition.

## 1 Introduction

This paper addresses a Finite State Machine (FSM) decomposition/synthesis problem loosely stated as follows. Replace a given input/output specification FSM by two FSMs, one of which is known and called a context FSM, another has to be determined; component FSMs communicate with the environment by external channels (interfaces) and with each other by non-observable internal channels. Replacement is considered legal, i.e., decomposition is achieved, if the two FSMs do not expose on external channels a behaviour illegal with respect to the specification FSM.

On one hand, the problem could be viewed as a generalization of the classical supervisory control problem [9] to open loop systems, since the environment and internal channels are absent at least in its basic setting. On the other hand, it is a specialization of the problem of solving equations over languages [15] to equations over input/output FSMs, since languages associated with input/output FSMs constitute a special class of regular languages. We refer to [15, 14, 11, 3, 1, 5, 9] for a survey of the literature.

In [15] we provided a formula defining the largest solution as the complement of the language obtained as a composition of the given component language and the complement of the language to be decomposed (the specification). This formula completely characterizes the solutions of a language inequation in the sense that a language is a solution of a given language inequation if and only if it is a sublanguage of the largest solution. If the composition of the largest solution of the inequality with the context is equal to the specification then it is the largest solution of the language equation; however not each sublanguage of the largest solution is a solution of the language equation (the complete characterization of such sublanguages is still an open problem). Furthermore, if a solution is wanted that corresponds to the language of an FSM, the obtained language must further be trimmed to enforce prefix-closure and coupling inputs with outputs, see [15]. If, moreover, such a solution should correspond to a complete FSM, where transitions are specified for all inputs in each state (inputs are never blocked), the resulting input/output FSM language has to be further trimmed to obtain an (input) progressive language. This, unfortunately, does not guarantee that the obtained solution is compositionally progressive, i.e., such that combined with the context FSM does not block inputs that can occur in the specification FSM. Compositionally progressive solutions of input/output FSM equations are studied in this paper.

Motivating contexts can be found in solving equations for synthesis/resynthesis of sequential logic, where it is assumed that hardware must be progressive [8]. Another application can be found in software wrapping, a technique in which an interface is created around an existing piece of software, providing a new view of the software to external systems, objects, or users. The problem of determining a wrapper can formally be cast as equation solving. The point here is that any software is normally progressive and should never block.

While the solutions to a general language inequality have been completely characterized, characterizing all compositionally progressive solutions of an input/output FSM equation is a non-trivial open problem. This may be explained at least by the facts that completeness of an input/output FSM and input progressiveness of the language associated with it do not always imply each other, and the union of two compositionally progressive FSMs may include an FSM that is not compositionally progressive. To completely solve the problem, one needs to first investigate FSM equations for each of the two different communication paradigms, synchronous and parallel: languages allow both types of composition [15], while input/output FSMS interact either synchronously or asynchronously. Both paradigms have useful, though different, target applications, e.g., synchronous composition of FSMs models the synchronous connection of sequential circuits. In this paper, we investigate synchronous FSM equations and contribute to solving the problem by providing algorithms to compute a largest compositionally progressive solution such that any extension of it is no longer a compositionally progressive solution, while every compositionally progressive solution is still contained in it. At the same time, the algorithms allow that the obtained largest compositionally progressive solution contains also solutions that are not compositionally progressive, as we later demonstrate.

The contributions of this work can be summarised as follows. It is shown that if a synchronous FSM equation has a compositionally progressive solution, then the equation has a largest compositionally progressive solution. If a largest solution is compositionally progressive, then it is a largest compositionally progressive solution. In the case when a largest solution is not compositionally progressive, a procedure is needed for iterative removal of illegal input-output strings that violate the property, until what is left is compositionally progressive or is empty (there is no such solution). Two variations of such a procedure are proposed in this paper. One is based on deleting all illegal strings from a largest solution (it uses complementation and therefore determinization at each step of the iteration). Another one is based on splitting and subsequent removal of bad states of a largest solution aiming at deleting only illegal strings and keeping compositionally progressive strings (the latter procedure uses determinization once in the state splitting step, but does not appeal to complementation in the iteration removing bad states). The results are non-trivial, as a largest compositionally progressive FSM solution is not necessarily a submachine of a largest FSM solution.

Related work addressed the same problem for parallel equations over automata [7, 2, 4]. Kumar *et al.* [7] solved the problem over finite automata, under the assumptions that the unknown has no external actions and that all the states of the automata are accepting. The authors define an automaton as a solution to the equation if the automaton is a solution and when combined with the context does not block any action available in the specification (called progressive). The authors show that a solvable equation always has a largest solution, i.e., an equation has a progressive solution if and only the equation has a largest progressive solution. An algorithm for deriving a largest progressive solution is given, based on splitting states of a chaos automaton (over the alphabet of the unknown), combining the split chaos automaton with the context and then deleting states from the split chaos automaton that accept strings that violate the specification. Each state of the split chaos automaton is such that it accepts either only sequences that are in a progressive solution (progressive state) or none of the accepted sequences is in a progressive solution.

The work by Kumar *et al.* [7] was generalized by K.El-Fakih and co-authors in [2, 4] to the case when the unknown has external actions and the automata may have non-accepting states, with the goal of deriving a largest progressive reduction of a given non-progressive solution. The latter task is non-trivial because the number of different reductions is infinite in general. Moreover, K.El-Fakih *et al.* propose a method for a complete characterization of the reductions of a largest non-progressive solution which are progressive solutions to a parallel automata equation.

However, the case of synchronous composition of FSMs is different from the parallel composition of automata. The actions of an FSM are partitioned into inputs and outputs and a compositionally progressive solution is required not to block inputs of the specification, while not caring about outputs when the specification is a non-deterministic FSM. Another difference is due to the semantics of the synchronous and parallel composition operators, because the synchronous composition operator processes actions in all channels simultaneously while the parallel composition operator processes an action of a single channel at a time.

This paper is organized as follows. Basic definitions about equations over languages and FSMs are provided, respectively, in Sec. 2 and Sec. 3. Compositionally progressive solutions are introduced in Sec. 4. An algorithm that deletes strings that are not compositionally progressive strings is given in Sec. 5. A different algorithm based on splitting and subsequent removal of states is presented in Sec. 6.

This paper is self-contained, but the reader is referred to [15] for an in-depth exposition of language equations.

## 2 Synchronous Equations over Languages

### 2.1 Languages and Automata

We remind the notions of substitution and homomorphism of languages, referring to a standard textbook for the basic definitions [6]. A **substitution** $f$ is a mapping of an alphabet $\Sigma$ to subsets of $\Delta^\star$ for some alphabet $\Delta$. 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 $\Sigma$. Two useful operations on languages are:

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

**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*

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

**Definition 2.3** *A* **finite automaton** *(FA) is defined as a 5-tuple $F = \langle S, \Sigma, \Delta, r, Q\rangle$. $S$ represents the finite state space, $\Sigma$ represents the finite alphabet, and $\Delta \subseteq \Sigma \times S \times S$ is the next state relation, such that $n \in S$ is a next state of present state $p \in S$ on symbol $i \in \Sigma$ if and only if $(i, p, n) \in \Delta$. The initial or reset state is $r \in S$ and $Q \subseteq S$ is the set of final or accepting states. A variant of FAs allows the introduction of $\epsilon$-moves, meaning that $\Delta \subseteq (\Sigma \cup \{\epsilon\}) \times S \times S$.*

*The next state relation can be extended to have as argument strings in $\Sigma^*$ (i.e., $\Delta \subseteq \Sigma^* \times S \times S$) as follows: $(\rho i, s, s'') \in \Delta$ if and only if there exists $s' \in S$ such that $(\rho, s, s') \in \Delta$ and $(i, s', s'') \in \Delta$.*

*A string $x$ is said to be* **accepted** *by the FA $F$ if there exists a sequence of transitions corresponding to $x$ such that there is a state $r' \in Q$ for which $\Delta(x, r, r')$. The* **language** *accepted by $F$, designated $L_r(F)$, is the set of strings $\{x \mid \exists r' \in Q \; [\Delta(x, r, r')]\}$. The language accepted or* **recognized** *by $s \in S$, denoted $L_r(F|s)$ or $L_r(s)$ when $F$ is clear from the context, is the set of strings $\{x \mid \Delta(x, r, s)\}$.*

*If for each present state $p$ and symbol $i$ there is at least one next state $n$ such that $(i, p, n) \in \Delta$, the FA is said to be* **complete***.*

*An FA is a* **deterministic finite automaton** *(DFA) if for each present state $p$ and symbol $i$ there is exactly one next state $n$ such that $(i, p, n) \in \Delta$. The relation $\Delta$ can be replaced by the next state function $\delta$, defined as $\delta : \Sigma \times S \to S$, where $n \in S$ is the next state of present state $p \in S$ on symbol $i \in \Sigma$ if and only if $n = \delta(i, p)$. An FA that is not a DFA is a* **non-deterministic finite automaton** *(NDFA).*

*A string $x$ is said to be* **accepted** *by the DFA $F$ if $\delta(x, r) \in Q$. The* **language** *accepted by $F$, designated $L_r(F)$, is the set of strings $\{x \mid \delta(x, r) \in Q\}$. The language accepted or* **recognized** *by $s \in S$, denoted $L_r(F|s)$ or $L_r(s)$ when $F$ is clear from the context, is the set of strings $\{x \mid \delta(x, r) = s\}$.*

**Definition 2.4** *Given a finite automaton $F = \langle S, I \times O, \Delta, r, Q\rangle$, state $p \in S$ is $I$-***complete** *if and only if, for each $i \in I$, there exist $n \in S$ and $o \in O$ such that $(io, p, n) \in \Delta$. A finite automaton is $I$-complete if each state is $I$-complete.*

A DFA is complete. The language of a DFA over $I \times O$ is $I$-progressive.

The languages associated with finite automata are the regular languages, defined by means of regular expressions, by Kleene's theorem [6]. Regular languages are closed under union, concatenation, complementation (the complement of language $L$ is denoted by $\overline{L}$), intersection and substitution [6] (therefore they are closed under projection and lifting, that are instances of substitution). The equivalence of regular expressions and finite automata is shown by matching each operation on regular expressions with a constructive procedure that yields the finite automaton of the result, given the finite automata of the operands. For the most common operations (union, concatenation, complementation, intersection) see [6]. Two finite automata are equivalent if and only if they accept the same language and state minimization yields a unique automaton associated to a given language [6]. Here we report from [15] the constructions for the less known operations of projection and lifting:

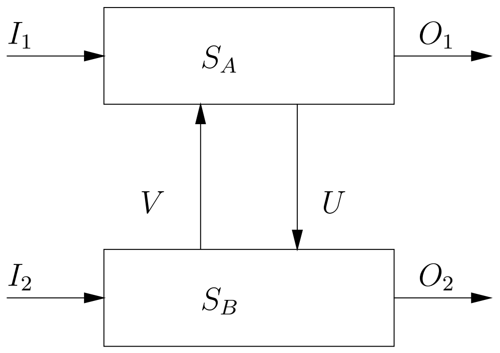

Figure 1: Composition topology.

**projection** **($\downarrow$)** Given FA $F$ that accepts language $L$ over $X \times V$, FA $F'$ that accepts language $L_{\downarrow V}$ over $X$ is obtained from $F$ by the following procedure:

replace each edge $((x, v), s, s')$ by the edge $(x, s, s')$.

**lifting** **($\uparrow$)** Given FA $F$ that accepts language $L$ over $X$, FA $F'$ that accepts language $L_{\uparrow V}$ over $X \times V$ is obtained from $F$ by the following procedure:

replace each edge $(x, s, s')$ by the edges $((x, v), s, s'),\ \forall v \in V$.

*In the sequel often we will not make a distinction between a regular language $L$ and the (determinized and minimized) finite automaton $F(L)$ that recognizes $L$, naming both of them by $L$.*

## 2.2 Synchronous Composition of Languages

Consider two systems $S_A$ and $S_B$ with associated languages $A$ and $B$. The systems communicate with each other by a channel $U$ and with the environment by channels $I$ and $O$, as in the topology of Fig. 1. We introduce a composition operator that describes the language associated to the external behaviour of the composition of $S_A$ and $S_B$. We assume the same order $I \times U \times O$ in the languages $(L_1)_{\uparrow O}$ and $(L_2)_{\uparrow I}$ (languages over product alphabets are defined up to permutations of component alphabets).

**Definition 2.5** *Given alphabets $I, U, O$, language $L_1$ over $I \times U$ and language $L_2$ over $U \times O$, the* **synchronous product** *of languages $L_1$ and $L_2$ is the language $(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}$.*

**Definition 2.6** *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 $[(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}]_{\downarrow I \times O}$, defined over $I \times O$, which is denoted by $L_1 \bullet_{I \times O} L_2$ and is abbreviated as $L_1 \bullet L_2$.*

**Definition 2.7** *Given a language $A$ over alphabet $I \times U$, a language $B$ over alphabet $U \times O$ is* **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 Synchronous Language Equations

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

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

**Definition 2.8** *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 a* **solution** *of the inequality $A \bullet X \subseteq C$ (of the equation $A \bullet X = C$) if and only if $A \bullet B \subseteq C$ ($A \bullet B = C$). $B = \emptyset$ is the* **trivial solution**.

**Definition 2.9** *A solution $S$ is the* **largest solution** *of the inequality $A \bullet X \subseteq C$ (of the equation $A \bullet X = C$) if and only if it contains every solution, i.e., it is such that if $A \bullet B \subseteq C$ ($A \bullet X = C$) then $B \subseteq S$.*

**Proposition 2.1** *If the largest solution of the inequality $A \bullet X \subseteq C$ is also a solution of the equation $A \bullet X = C$, then it is the largest solution of the equation $A \bullet X = C$, otherwise the equation has no solution.*

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

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

$$\begin{array}{rcl}
A \bullet \{\alpha\} \subseteq C & \Leftrightarrow & \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I})_{\downarrow I \times O} \cap \overline{C} = \emptyset & \Leftrightarrow & \textit{by Prop. } A.1(a)\ \overline{C} = (\overline{C}_{\uparrow U})_{\downarrow I \times O} \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I})_{\downarrow I \times O} \cap (\overline{C}_{\uparrow U})_{\downarrow I \times O} = \emptyset & \Leftrightarrow & \textit{by Prop. } A.2(d) \textit{ since } ((\overline{C}_{\uparrow U})_{\downarrow I \times O})_{\uparrow U} = \overline{C}_{\uparrow U} \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U})_{\downarrow I \times O} = \emptyset & \Leftrightarrow & \\
A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U} = \emptyset & \Leftrightarrow & \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O} = \emptyset & \Leftrightarrow & \textit{by Prop. } A.2(d) \textit{ since } \{\alpha\}_{\uparrow I} = ((\{\alpha\}_{\uparrow I})_{\downarrow U \times O})_{\uparrow I} \\
(\{\alpha\}_{\uparrow I})_{\downarrow U \times O} \cap (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O} = \emptyset & \Leftrightarrow & \textit{by Prop. } A.1(a)\ (\{\alpha\}_{\uparrow I})_{\downarrow U \times O} = \{\alpha\} \\
\{\alpha\} \cap (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O} = \emptyset & \Leftrightarrow & \\
\alpha \notin (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O} & \Leftrightarrow & \\
\alpha \in \overline{(A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O}} & \Leftrightarrow & \\
\alpha \in \overline{A \bullet \overline{C}} & &
\end{array}$$

Therefore the largest solution of the language equation $A \bullet X \subseteq C$ is given by the language

$$S = \overline{A \bullet \overline{C}}. \tag{2}$$

□

The proof formalizes the fact that the largest solution is the complement of all forbidded strings, i.e., the strings that are accepted by the context $A$ but are not in the specification $C$.

**Definition 2.10** *A solution $S_{CP}$ of the inequality $A \bullet X \subseteq C$ is* **compositionally I-progressive** *if and only if $S_{CP}$ is compositionally $I$-progressive, i.e., $A_{\uparrow O} \cap S_{CP \uparrow I}$ is $I$-progressive.*

*A compositionally $I$-progressive solution $\tilde{S}_{CP}$ is the* **largest compositionally I-progressive solution** *of the inequality $A \bullet X \subseteq C$ if and only if it contains every compositionally $I$-progressive solution, i.e., it is such that if $A \bullet B \subseteq C$ and $B$ is compositionally $I$-progressive, then $B \subseteq \tilde{S}_{CP}$.*

We would like to obtain again a complete characterization of such solutions (as Th. 2.1 does for the largest solution), i.e., we would like to compute a set $S_{CP} \subseteq S$, such that it contains all and only the compositionally $I$-progressive solutions:

1. if $A \bullet B \subseteq C$ and $A_{\uparrow I_2 \times O_2} \cap B_{\uparrow I_1 \times O_1}$ is $I$-progressive, then $B \subseteq S_{CP}$;
2. if $B \subseteq S_{CP}$ and $B$ is $(I_2 \times U)$-progressive, then $A \bullet B \subseteq C$ and $A_{\uparrow I_2 \times O_2} \cap B_{\uparrow I_1 \cap O_1}$ is $I$-progressive.

However, the problem of characterizing all compositionally $I$-progressive solutions is a non-trivial open problem. In this paper we will work out part of the solution, by providing algorithms to compute a regular language $\tilde{S}_{CP}$ such that:

1. if $A \bullet B \subseteq C$ and $A_{\uparrow I_2 \times O_2} \cap B_{\uparrow I_1 \cap O_1}$ is $I$-progressive, then $B \subseteq \tilde{S}_{CP}$.

Of course we could take the largest solution $S$ as $\tilde{S}_{CP}$, but this would not be an interesting result, so to make the objective more interesting we look for a regular language $\tilde{S}_{CP}$ such that:

**P1** if $A \bullet B \subseteq C$ and $A_{\uparrow I_2 \times O_2} \cap B_{\uparrow I_1 \cap O_1}$ is $I$-progressive, then $B \subseteq \tilde{S}_{CP}$;

**P2** $\tilde{S}_{CP}$ is compositionally $I$-progressive.

In other words, in obtaining $\tilde{S}_{CP}$ we should not "take away" from $S$ too much, so that at the end every compositionally $I$-progressive solution is still contained in $\tilde{S}_{CP}$. We call $\tilde{S}_{CP}$ the *largest compositionally $I$-progressive solution*, because each regular language $L$ such that $L \not\subseteq \tilde{S}_{CP}$ is not compositionally $I$-progressive. However, it may happen that $\tilde{S}_{CP}$ contains also solutions that are not compositionally $I$-progressive, as Example 3.1 will show, when talking about regular languages corresponding to FSMs.

## 3 Synchronous Equations over FSM Languages

### 3.1 Languages of Finite State Machines

**Definition 3.1** *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 $o$ if and only if $(i, p, n, o) \in T$. State $r \in S$ represents the initial or reset state. An FSM is said to be* **trivial** *when $T = \emptyset$, denoted by $M_\epsilon$.*

**Definition 3.2** *If at least one transition is specified for each present state and input pair, an FSM is said to be* **complete***, otherwise it is* **incomplete** *or* **partial***; similarly, a* **complete state** *has at least one transition for each input, otherwise it is an* **incomplete state** *or an* **incompletely specified state** *or an* **undefined state** *(under some inputs).*

**Definition 3.3** *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 3.4** *A* **deterministic FSM** *(DFSM) is an FSM where for each pair $(i, p) \in I \times S$, there is at most one next state $n$ and one output $o$ such that $(i, p, n, o) \in T$, i.e., there is at most one transition from $p$ under $i$. An FSM that is not a DFSM is a* **non-deterministic finite state machine** *(NDFSM).*

**Definition 3.5** *An NDFSM is an* **observable FSM** [10] *(OFSM), a.k.a.* **pseudo-nondeterministic FSM** *(PNDFSM), if for each triple $(i, p, o) \in I \times S \times O$, there is at most one state $n$ such that $(i, p, n, o) \in T$.*

**Definition 3.6** *A complete FSM is said to be of* **Moore** *type if $(i, p, n, o) \in T$ implies that for all $i'$ there is $n'$ such that $(i', p, n', o) \in T$.*

This definition is more general than the usual one to apply also to non-deterministic FSMs.

**Note** *In this paper FSMs are assumed to be complete and pseudo non-deterministic, unless otherwise stated. It is always possible to convert a general NDFSM into a PNDFSM by subset construction.*

**Definition 3.7** *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 $((i, o), s, s') \in \Delta$ if and only if $(i, s, s', o) \in T$. The language accepted by $F(M)$ when started at state $s \in S$ is denoted by $L_s(F(M))$ ($s$ does not need to be an initial state), abbreviated as $L_s(M)$ or $L(M)$, when $s$ is the initial state.*

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

**Definition 3.8** *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**[1].

**Definition 3.9** *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 $t \preceq s$, if and only if $L_t(M_B) \subseteq L_s(M_A)$. States $t$ and $s$ are* **equivalent states***, written $t \cong s$, if and only if $t \preceq s$ 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 $M_A$, $M_B \preceq M_A$, if and only if $r_{M_B}$, 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., $M_A \simeq M_B$.*

Two equivalent states do not accept the same language (as accepting states), but generate the same language (as initial states).

## 3.2 Synchronous Composition of FSMs

The synchronous composition of two FSMs models the synchronous connection of sequential circuits. In general, the composition of FSMs is a partially specified function from pairs of FSMs to an FSM.

Consider a pair of FSMs: $M_A$ with input alphabet $I_1 \times V$ and output alphabet $U \times O_1$, $M_B$ with input alphabet $I_2 \times U$ and output alphabet $V \times O_2$. We define a synchronous composition operator $\bullet$ that associates to a pair of FSMs $M_A$ and $M_B$ another FSM $M_A \bullet M_B$, with external input alphabet $I = I_1 \times I_2$ and external output alphabet $O = O_1 \times O_2$. Henceforth, unless otherwise stated, we set $I = I_1 \times I_2$ and $O = O_1 \times O_2$.

**Definition 3.10** *The* **synchronous composition of FSMs** *$M_A$ and $M_B$ yields the reduced observable FSM denoted by $M_A \bullet_{I \times O} M_B$ (abbreviated as $M_A \bullet M_B$), whose language is*

$$[L(M_A)_{\uparrow I_2 \times O_2} \cap L(M_B)_{\uparrow I_1 \times O_1}]_{\downarrow I \times O}.$$

The FSM $M_A \bullet M_B$ can be complete or partial. In practice, we convert from the FSMs $M_A$ and $M_B$ to the automata accepting their FSM languages, operate on them and then convert back from the resulting automaton to a reduced and observable FSM. To obtain an observable FSM, a determinization step may be required.

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

## 3.3 Solutions of Synchronous FSM Equations

Given alphabets $I_1, I_2, U, V, O_1, O_2$, an FSM $M_A$ over inputs $I_1 \times V$ and outputs $U \times O_1$, and an FSM $M_C$ over inputs $I_1 \times I_2$ and outputs $O_1 \times O_2$, consider the FSM inequality $M_A \bullet M_X \preceq M_C$, whose unknown is an FSM $M_X$ over inputs $I_2 \times U$ and outputs $V \times O_2$. The FSM $M_A$ is a known component machine (the context), the FSM $M_C$ is the specification of the overall system and the equation defines the FSMs that in place of $M_X$ let the synchronous composition be a reduction of the specification $M_C$.

**Definition 3.11** *FSM $M_B$ is a* **solution** *of the inequality $M_A \bullet M_X \preceq M_C$ (of the equation $M_A \bullet M_X \simeq M_C$), where $M_A$ and $M_C$ are FSMs, if and only if $M_A \bullet M_B \preceq M_C$ ($M_A \bullet M_B \simeq M_C$).*

**Definition 3.12** *A solution $M_S$ of the inequality $M_A \bullet M_X \preceq M_C$ (of the equation $M_A \bullet M_X \simeq M_C$), where $M_A$ and $M_C$ are FSMs, is a* **largest solution** *if and only if it contains every solution, i.e., it is such that if $M_A \bullet M_B \preceq M_C$ ($M_A \bullet M_X \simeq M_C$) then $M_B \preceq M_S$.*

An FSM largest solution is not unique because there may be different FSMs that accept the same language and thus satisfy a given inequality or equation.

**Proposition 3.1** *If a largest solution of the inequality $M_A \bullet M_X \preceq M_C$ is also a solution of the equation $M_A \bullet M_X \simeq M_C$, then it is a largest solution of the equation $M_A \bullet M_X \simeq M_C$, otherwise the equation has no solution.*

Operating on the FSM languages of the given FSMs, one applies Th. 2.1 to find the largest language solution and then restricts the latter to be an FSM language (enforcing prefix-closure) [15]. Each FSM that accepts this largest language solution is a largest FSM solution. Every FSM solution is a reduction of a largest FSM solution, i.e., the behavior of any FSM that is a solution is contained in a largest solution of the FSM inequality, and also any reduction of a largest FSM solution is a solution of the FSM inequality; however, not every reduction of a largest FSM solution is a solution of the FSM equation. The solution procedure presented in [15] returns a PNDFSM as a largest FSM solution.

All the operations required to compute the solutions of FSM equations are carried out on the automata corresponding to their FSM languages, according to the constructions mentioned at the end of Sec. 2.1.

## 3.4 Compositionally Progressive Solutions of Synchronous FSM Equations

In this paper we study FSM solutions $M_B$ such that the language $L(M_A)_{\uparrow I_2 \times O_2} \cap L(M_B)_{\uparrow I_1 \times O_1}$ is $I$-progressive. Such solutions, when combined with the context, do not block any input that may occur in the specification, so they are of practical use.

**Definition 3.13** *A solution $M_{S_{CP}}$ of the inequality $M_A \bullet M_X \preceq M_C$, where $M_A$ and $M_C$ are FSMs, is a* **compositionally $I$-progressive solution** *if and only if $L(M_A)_{\uparrow I_2 \times O_2} \cap L(M_{S_{CP}})_{\uparrow I_1 \times O_1}$ is $I$-progressive, where $I = I_1 \times I_2$.*

*A solution $M_{\tilde{S}_{CP}}$ of the inequality $M_A \bullet M_X \preceq M_C$, where $M_A$ and $M_C$ are FSMs, is a* **largest compositionally $I$-progressive solution** *if and only if it contains every compositionally $I$-progressive solution, i.e., it is such that if $M_A \bullet M_B \preceq M_C$ and $M_B$ is a compositionally $I$-progressive solution then $M_B \preceq M_{\tilde{S}_{CP}}$.*

Example 3.1 provides some intuition to the fact that, since generally the number of reductions is infinite, the problem of characterizing all compositionally $I$-progressive solutions is a non-trivial open problem.

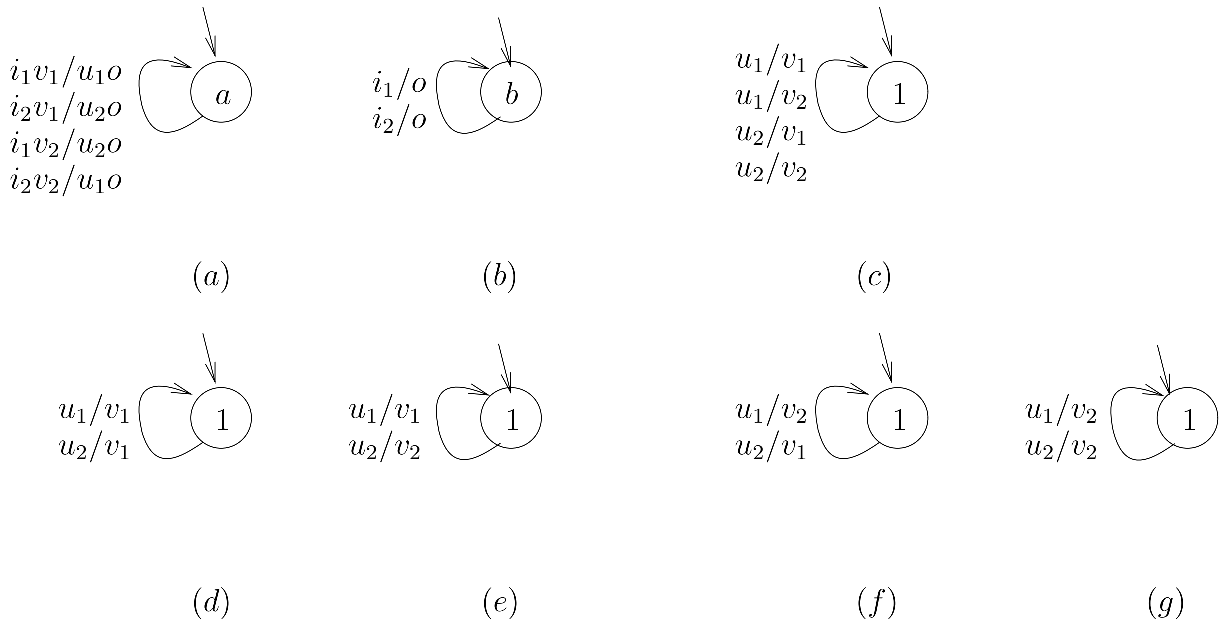

Figure 2: Illustration of Example 3.1. (a) Context FSM $M_A$; (b) Specification FSM $M_C$; (c) Largest FSM solution $M_S$; (d) DFSM solution $M_{S_1}$, $U$-progressive and compositionally $I$-progressive; (e) DFSM solution $M_{S_2}$, $U$-progressive, but not compositionally $I$-progressive; (f) DFSM solution $M_{S_3}$, $U$-progressive, but not compositionally $I$-progressive; (g) DFSM solution $M_{S_4}$, $U$-progressive and compositionally $I$-progressive.

**Example 3.1** *Fig. 2 shows an instance of FSM language equation, where the FSMs for the context, specification and largest solution are given, respectively, in Figs. 2(a)-(c), whereas Figs. 2(d)-(g) portray some deterministic FSMs (DFSMs) contained in a largest solution. All these solutions are $U$-progressive, but two of them are not compositionally $I$-progressive: e.g., the DFSM in Fig. 2(e) is not compositionally $I$-progressive, because its composition with $M_A$ yields an FSM undefined under input $i_2$. We claim that largest solution $M_S$ in Fig. 2(c) is a largest compositionally $I$-progressive solution, however - as pointed out - it includes solutions that are not compositionally $I$-progressive: by removing any transition from $M_S$ we would miss some compositionally $I$-progressive solutions. The key fact is that the union of two compositionally $I$-progressive solutions may include a solution that is not compositionally $I$-progressive, e.g., the union of the FSMs in Figs. 2(d) and (g) contains the FSMs in Figs. 2(e) and (f).*

A sufficient condition to guarantee that a solution $M_B$ is compositionally $I$-progressive is that $M_B$ satisfies the Moore property. If a largest solution has a complete reduction that is a Moore FSM, then this reduction is a compositionally $I$-progressive solution. However, the equation may have a compositionally $I$-progressive solution even if there is no Moore solution.

If a largest solution is compositionally $I$-progressive, then it is the largest compositionally $I$-progressive solution. Otherwise, input-output strings that violate the property must be deleted from those accepted by a largest solution, until what is left is compositionally $I$-progressive or it is empty (there is no such solution). The trimming procedure is not trivial, because in general a largest solution has an infinite number of input-output strings. An alternative way is to delete states that are not $I$-progressive from a largest solution. However, we should not delete states directly. To understand the problem, consider the example (from a publication in Russian by S. Zharikova et al. [13]) of context and specification in Figs. 5(a) and 5(b), whose solution is shown in Fig. 5(c). The largest solution in Fig. 5(c) is not compositionally $I$-progressive, because state $(b, b_1a_2)$ is not defined under input $i_2$ (see Fig. 5(d)). By deleting the state $b_1a_2$ from $S$ to avoid that state $(b, b_1a_2)$ is generated in the composition, we would miss the reduction $\hat{S}$ of $S$ shown in Fig. 5(e), that is instead compositionally $I$-progressive (as demonstrated by the composition in Fig. 5(f)). The fact is that in the largest solution in Fig. 5(c) the state $(b_1, a_2)$ can be reached from the initial state via either the string $v_1u_1$ or the string $v_1u_2v_1u_1$, and we will see in Sec. 4 that these two strings behave differently with respect to the property of the composition being progressive.

Related work has been reported in the Ph.D. dissertation by M. Vetrova at Tomsk (available only in Russian) [12] about the special cases of the series and controller's topologies, for which it has been computed $\tilde{S}_{CP}$ that contains all and only compositionally $I$-progressive solutions.

## 3.5 Complete FSMs vs. Progressive Languages

We show in this section that the notions that an FSM $M$ is complete and that its language $L((F(M))$ is $I$-progressive turn out to be equivalent when the automaton $F(M)$ associated to $M$ is deterministic. This justifies the way in which FSM completeness, a structural property, and language progressiveness, a behavioural property, are used where appropriate in the algorithms to compute a largest compositionally progressive solution, introduced in Sec. 4.

An FSM is complete when at all states a transition is defined under every input. A language over alphabet $I \times O$ is $I$-progressive if, for each word $\alpha$ in the language and any $i \in I$, there is $o \in O$, such that $\alpha(i, o)$ is in the language. From Defn. 3.7 in Sec. 3.1, to each FSM is associated a language. It is natural to ask whether the language of a complete FSM is always $I$-progressive and vice versa, whether an FSM whose language is $I$-progressive must be complete.

The answers are:

1. Given a complete FSM, its associated language is $I$-progressive; the reason is that each word in the language reaches an accepting state from which the word can be extended under every input to another accepting state.

2. An FSM whose language is $I$-progressive does not need to be complete, as it is shown in Ex. 3.2.

3. A PNDFSM whose language is $I$-progressive is complete; the reason is that each word in the language reaches a *unique* accepting state (the underlying automaton of a PNDFSM is deterministic) and, since the associated language is $I$-progressive, from that reached state there must be a transition under every possible input.

Notice that in this paper we study the product $L(M_A)_{\uparrow O} \cap L(M_B)_{\uparrow I}$, where $M_B$ is a PNDFSM (by construction we obtain a PNDFSM as largest solution of a synchronous FSM equation, see [15]), and $M_A$ is a PNDFSM by the assumption in Sec. 3.1; therefore the composition of $M_A$ and $M_B$ is a PNDFSM too, for which the notions of progressive language and complete automaton coincide, as argued above.

The two following examples illustrate the previous statements.

**Example 3.2** *Fig. 3 shows an example of FSM whose language is $I$-progressive without the FSM being complete. The FSM is shown in Fig. 3(d), and it is neither a DFSM nor a PNDFSM, because from state $a1$ under $i_1/o_1$ two different transitions can be taken. The FSM is not complete, because at state $c3$ there is no transition under input $i_2$. The language of the FSM is $I$-progressive because $i1o1$ can be extended both with $i_1$ and $i_2$ by taking the transition to state $b2$ and then the self-loops.*

*By determinizing the underlying automaton one gets the FSM in Fig. 3(e) that is a PNDFSM and so is complete. Notice that the notion of $I$-progressive language is functional, whereas FSM completeness is a matter of its representation: different FSMs may be chosen to represent the same language.*

*Since in this paper we are discussing compositionally progressive languages, we show that the FSM in Fig. 3(d) could have been obtained by starting from the FSMs in Fig. 3(a) and (b), and then computing their synchronous composition as in Fig. 3(c), and finally projecting the composition over the alphabet $I \times O$ as in Fig. 3(d).*

**Example 3.3** *Fig. 4 shows two FSMs that are not complete and whose associated languages are not $I$-progressive. The FSM in Fig 3.3(a) is a DFSM that is not complete at state $s3$ (no transition under input $i_2$) and whose language is not $I$-progressive (the word $i_1 o_1$ cannot be extended with input $i_2$); the same holds for the PNDFSM in Fig 3.3(b) ($s3$ is not complete under $i_2$ and the word $i_1 o_2$ cannot be extended with input $i_2$). Since we argued that a complete FSM has an $I$-progressive language, it is expected that when the language is not $I$-progressive the FSM cannot be complete.*

## 4 Computation of Largest Compositionally Progressive FSM Solutions

In this section we remind first how to compute progressive FSM solutions and then we set the stage for two procedures to compute computationally progressive FSM solutions, that will be presented in detail, respectively, in Sec. 5 and 6.

Given a regular language $L$ over alphabet $I \times O$, one can build $L^{FSM}$, the largest subset of $L$ that is the language of an FSM over input alphabet $I$ and output alphabet $O$; furthermore, by Proc. 4.1, one can build $Prog(L^{FSM})$, the largest subset of $L^{FSM}$ that is the language of a complete FSM. Refer to [15] for a detailed exposition.

**Procedure 4.1** *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 $Prog(L^{FSM})$.

We are interested to compute the subset of compositionally $I$-progressive solutions $L(M_B)$, i.e., those such that $L(M_A)_{\uparrow I_2 \times O_2} \cap L(M_B)_{\uparrow I_1 \times O_1}$ is an $I$-progressive FSM language. Thus the composition (after projection onto $I \times O$) is the language of a complete FSM over inputs $I_1 \times I_2$ and outputs $O_1 \times O_2$. If $S^{FSM}$ is compositionally $I$-progressive, then $S^{FSM}$ is the largest compositionally $I$-progressive solution of the equation. Otherwise, let $cProg(S^{FSM})$ be the largest compositionally $I$-progressive subset of $S^{FSM}$. Conceptually $cProg(S^{FSM})$ is obtained from $S^{FSM}$ by deleting each string $\alpha$ such that, for some $i \in I$, there is no $(u, v, o) \in U \times V \times O$ for which it holds $\alpha\ (i, u, v, o) \in A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{\uparrow I_1 \times O_1}$.

Given the largest solution $S$, a string $\alpha \in S$, is called *compositionally progressive* if there exists a compositionally progressive solution including the string (in other words, there exists a compositionally

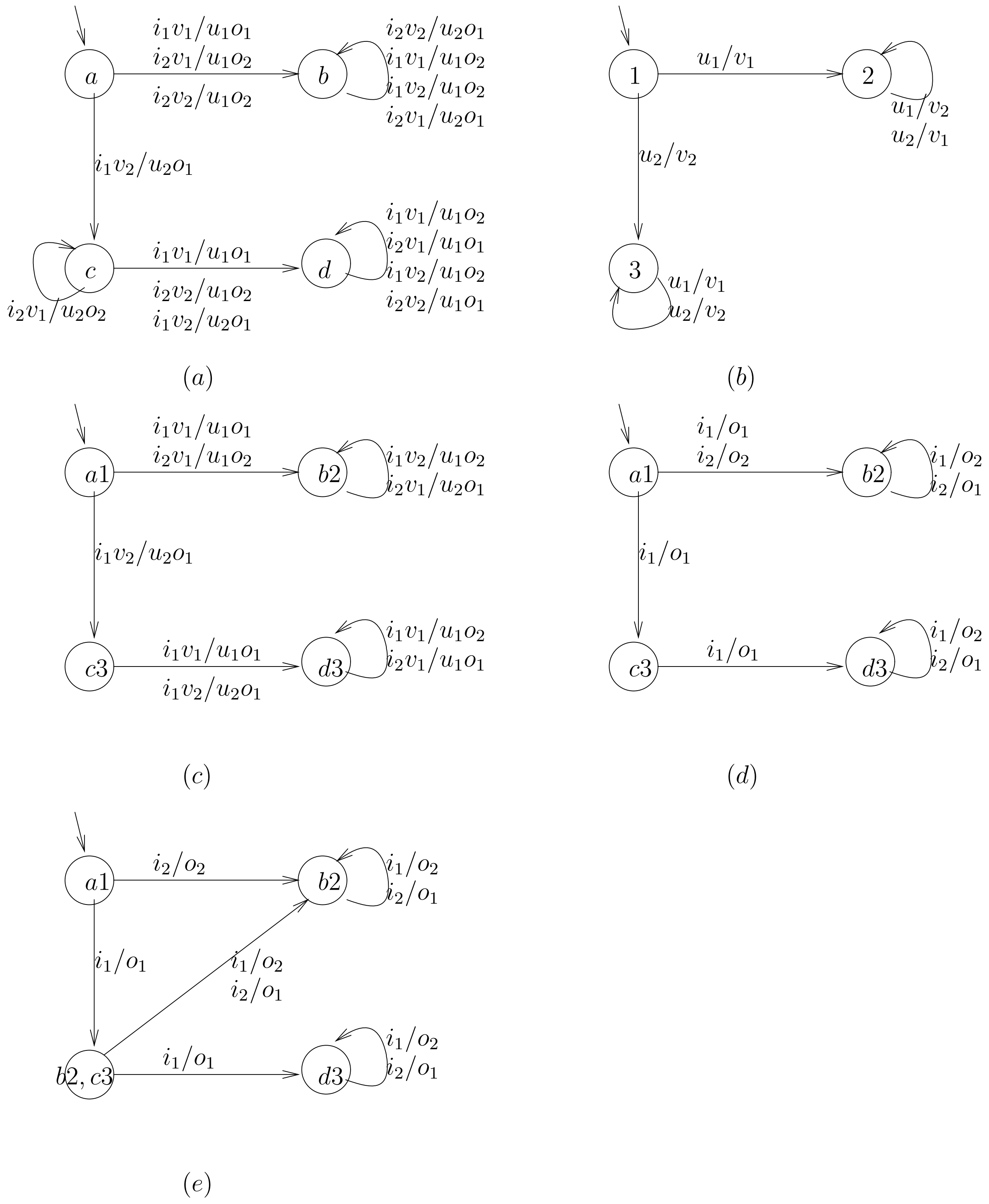

Figure 3: Illustration of Example 3.2. (a) FSM $M_A$; (b) FSM $M_S$; (c) Product FSM $M_A \cap M_{S_{\uparrow I \times O}}$ (language not $I$-progressive and FSM not complete); (d) Synchronous composition $M_A \bullet_{I \times O} M_S$ (language $I$-progressive, but FSM not complete); (e) Determinization (of underlying automaton) of $M_A \bullet_{I \times O} M_S$ (language $I$-progressive and FSM complete).

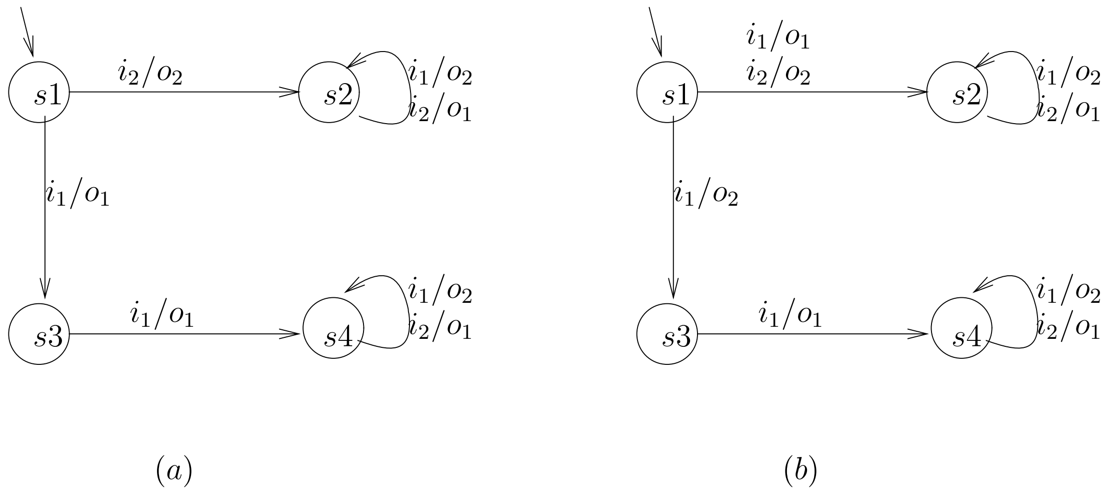

Figure 4: Illustration of Example 3.3. (a) Incomplete DFSM whose language is not $I$-progressive; (b) Incomplete PNDFSM whose language is not $I$-progressive.

progressive solution whose automaton accepts $\alpha$). For instance, in Fig. 5(c), $v_1 u_2 v_1 u_1 \in S$ is compositionally progressive, as witnessed by the compositionally progressive solution in Fig. 5(e). A string $\alpha \in S$, that is not compositionally progressive, i.e., no solution including the string is compositionally progressive, is called *compositionally non-progressive*. We say that such a string *induces* a compositionally non-progressive solution. For instance, in Fig. 5(c), $v_1 u_1 \in S$ is not compositionally progressive.

This suggests two options: either we delete all compositionally non-progressive strings from the largest solution or we "split" states of the largest solution into equivalent states in such a way that each state of the modified automaton $S_{split}$ is reachable from the initial state either only by compositionally progressive strings or only by compositionally non-progressive strings. A sub-automaton of $S_{split}$ restricted to the states reachable by compositionally progressive strings yields the largest compositionally progressive solution.

The following procedures tell how to compute $cProg(S^{FSM})$. The one based on deleting compositionally non-progressive strings from the largest solution will be presented in Sec. 5. It requires a difficult proof of termination (potentially there are infinite strings to be deleted). The approach based on state splitting is described in Sec. 6.

## 5 Computation by String Removal

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

**Procedure 5.1** *Input: Largest prefix-closed solution $S^{FSM}$ of synchronous inequality $A \bullet X \subseteq C$ 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 $R^i$.

(b) Compute $T^i = S^i \setminus B^i$, where $B^i = (R^i \setminus Prog(R^i))_{\downarrow I_2 \times U \times V \times O_2}$.

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 5.1** *Proc. 5.1 returns the largest compositionally $I$-progressive (prefix-closed) solution, if it terminates.*

**Proof.** By construction, when it terminates, Proc. 5.1 returns a language $S^{i+1}$ that is a compositionally $I$-progressive solution.

It must be proved that $S^{i+1}$ is the largest compositionally $I$-progressive solution, i.e., that any compositionally $I$-progressive solution $L^{FSM}$ is contained in $S^{i+1}$.

The proof goes by induction. By construction $L^{FSM} \subseteq S^1$ ($L^{FSM}$ is a solution). Suppose by induction hypothesis that $L^{FSM} \subseteq S^i$ and let us prove that $L^{FSM} \subseteq S^{i+1}$. Since $L^{FSM} \subseteq S^i$, the language $R = A_{\uparrow I_2 \times O_2} \cap L^{FSM}_{\uparrow I_1 \times O_1}$ is a subset of $R^i = A_{\uparrow I_2 \times O_2} \cap S^i_{\uparrow I_1 \times O_1}$. As the language $R$ is $I$-progressive then it is a subset of $Prog(R^i)$, that is by construction the largest $I$-progressive subset of $R^i$, and so by logical implication $L^{FSM}$ does not intersect $(R^i \setminus Prog(R^i))_{\downarrow I_2 \times U \times V \times O_2}$. Therefore $L^{FSM}$ is contained in the largest FSM language that is a subset of $T^i = S^i \setminus (R^i \setminus Prog(R^i))_{\downarrow I_2 \times U \times V \times O_2}$, that is by definition $T^{i\ FSM} = S^{i+1}$. Note that a largest compositionally progressive FSM solution may not be a submachine of any largest FSM solution, whereas any largest FSM solution has a largest FSM solution that is a complete FSM as a submachine and a largest FSM solution that is a Moore FSM as a submachine (if such solutions exist) [15].

Finally if $S^{i+1} = \emptyset$, from the fact that any compositionally $I$-progressive solution $L^{FSM}$ must be contained in $S^{i+1}$, it follows that $L^{FSM} = \emptyset$, i.e., there is no compositionally $I$-progressive solution. □

**Example 5.1** *Given the FSMs $M_A$ and $M_C$ in Fig. 5(a)-(b), the largest language solution $S$ in Fig. 5(c) of the inequality $M_A \bullet M_X \preceq M_X$ is $U$-progressive, but not compositionally $I$-progressive, because $R^1 = A \cap S^1_{\uparrow I_1 \times O_1}$ in Fig. 5(d) is not $I$-progressive ($S^1 = S^{FSM}$), given that there is no transition under input $i_2$ from the state labeled by $(b, b_1 a_2)$. The steps of Proc. 5.1, Figs. 6(a)-(b) and 6(c)-(d), show the computations to find $S^2$; since $R^2 = A \cap S^2_{\uparrow I_1 \times O_1}$ in Fig. 6(e) is $I$-progressive, $S^2$ is compositionally $I$-progressive and $cProg(S^{FSM}) = S^2$. Finally $M_{S^2}$ in Fig. 6(f) is a largest compositionally $I$-progressive FSM solution.*

**Theorem 5.2** *Proc. 5.1 terminates.*

**Proof.** The proof is given in the Appendix A. □

The proof relies on 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

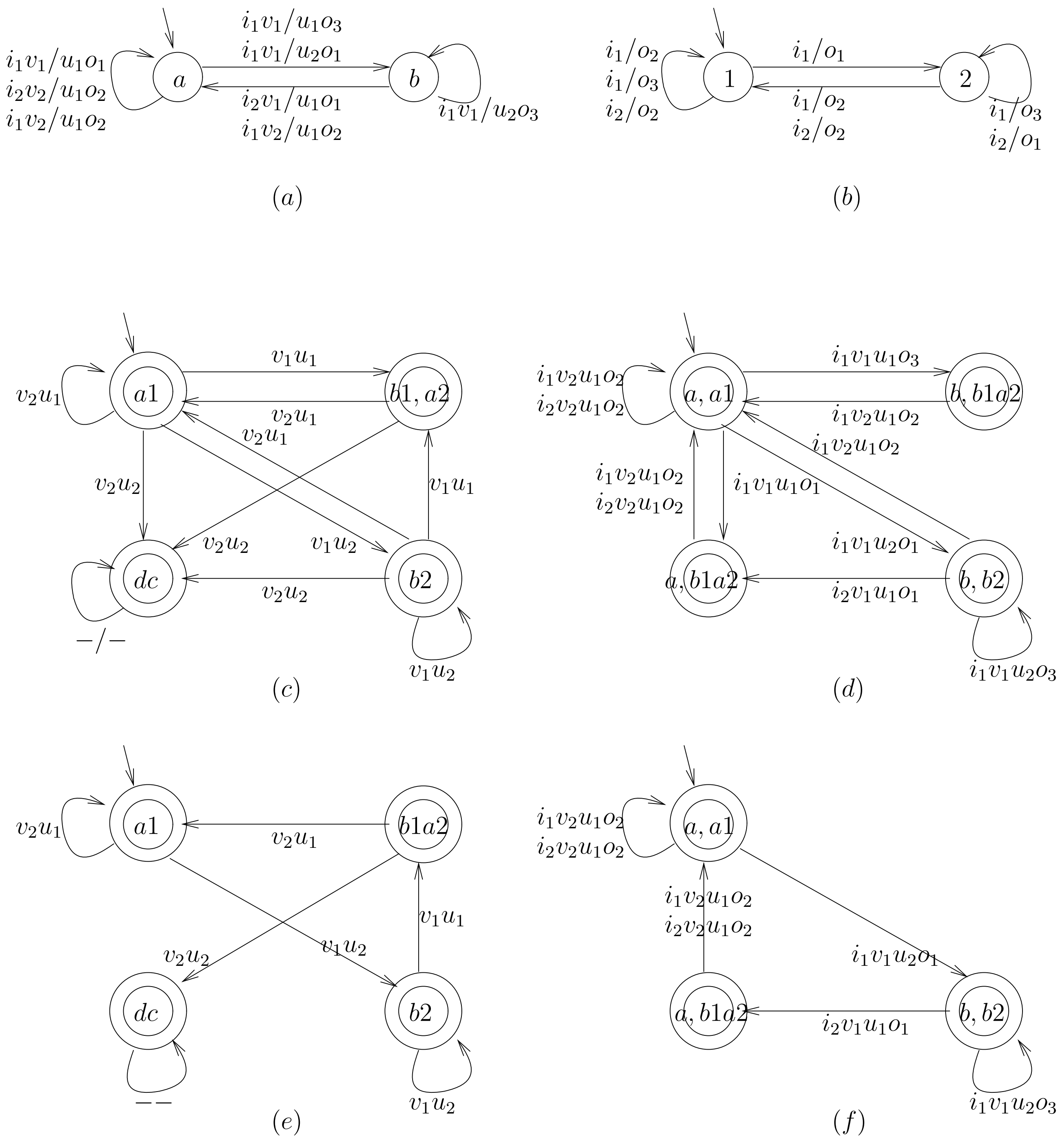

Figure 5: Illustration of Example 5.1. (a) FSM $M_A$; (b) FSM $M_C$; (c) Largest language solution $S = \overline{A \bullet \overline{C}}$; (d) $R^1 = A \cap S^1_{\uparrow I_1 \times O_1}$, with $S^1 = S^{FSM}$; (e) A particular solution $\hat{S}$; (f) $A \cap \hat{S}^1_{\uparrow I_1 \times O_1}$.

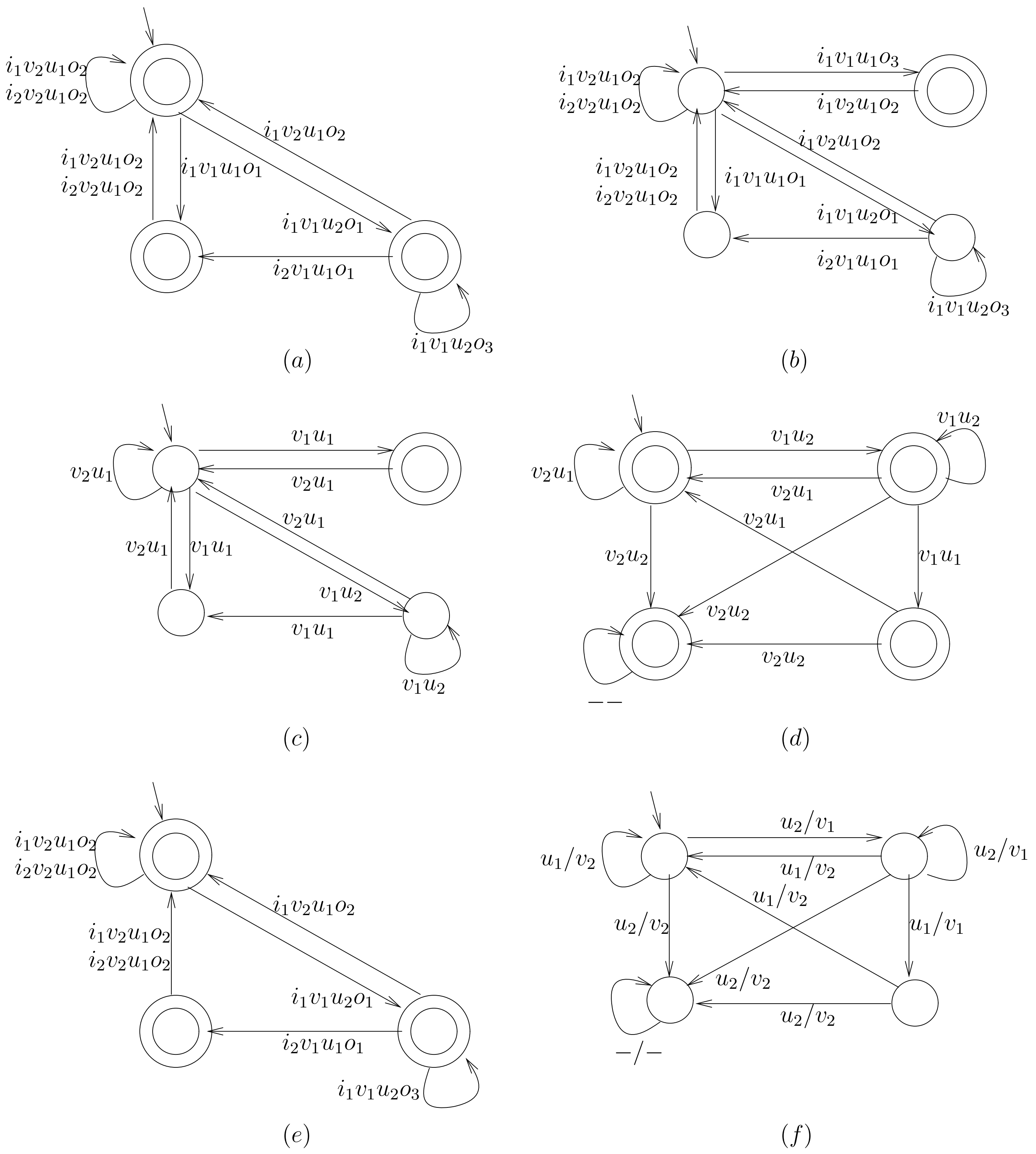

Figure 6: Illustration of Example 5.1. (a) $Prog(R^1)$; (b) $R^1 \setminus Prog(R^1)$; (c) $B^1 = (R^1 \setminus Prog(R^1))_{\downarrow V \times U}$; (d) $T^1 = S^1 \setminus B^1$; (e) $R^2 = A \cap S^2_{\uparrow I_1 \times O_1}$, with $S^2 = T^{1\ FSM}$; (f) Largest compositionally progressive FSM solution $M_{S^2}$.

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.

To provide some intuition about the proof 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 $L(q^1_j)$ [2]. 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 $R^1$:

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

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

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

- Language $R^1 \setminus \mathit{Prog}(R^1)$. 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 $B^1 = (R^1 \setminus \mathit{Prog}(R^1))_{\downarrow I_2 \times U \times V \times O_2}$. It is recognized by the NDFA $F^1_{B_1} = \langle S^1, I_2 \times U \times V \times O_2, \Delta^1, r^1, Q^1_{NP} \rangle$, 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 $R^2 = R^1 \cap (\overline{(R^1 \setminus \mathit{Prog}(R^1))_{\downarrow I_2 \times U \times V \times O_2}})_{\uparrow I_1 \times O_1} = R^1 \cap \overline{B^1}_{\uparrow I_1 \times O_1}$. To complement $B^1$, the NDFA of $B^1$ must be determinized yielding a DFA $F^1_{\hat{B}^1} = \langle \hat{S}^1, I_2 \times U \times V \times O_2, \hat{\Delta}^1, \hat{r}^1, \hat{Q}^1 \rangle$ 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 $\hat{q}^1_j = \{q^1_h \mid h \in H^1_j \subseteq J^1\}$ and $L(\hat{q}^1_j) = \bigcap_{h \in H^1_j \subseteq J^1} L(q^1_h)_{\downarrow I_2 \times U \times V \times O_2}$. Moreover, a *don't care* state $\{dc\}$ is added, which recognizes the language obtained as the complement of the union of the languages recognized by the states of the determinized automaton:

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

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

[2] By Def. 2.3, 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 $s$.

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 $q^2_j = (q^1_j, \hat{q}^1_p) \in Q^2,\ j \in J^2 = \{1, \ldots, |Q^2|\}$, recognizes the language $L(q^2_j)$ expressed in the following form: if $\hat{q}^1_p$ is not the *dc* state

$$L(q^2_j) = L(q^1_j) \ \cap \bigcap_{h \in H^1_j \subseteq J^1} (L(q^1_h)_{\downarrow I_2 \times U \times V \times O_2})_{\uparrow I_1 \times O_1},$$

otherwise if $\hat{q}^1_p$ is the *dc* state

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

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

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. A.2. We note that $H^1_j$ cannot be empty, since it corresponds to a non-empty subset of states of $R^1$, and that for the *dc* state we consider only non-empty subsets $K$.

## 6 Computation by State Splitting and Removal

Before stating formally algorithms and theorems, we explain intuitively the state splitting procedure. When we combine the context with the largest solution, we may reach a non-progressive state $as$ such that the projection of the language accepted by state $as$ is strictly contained in the language accepted by state $s$. Suppose that we delete state $as$ from the intersection and state $s$ from the solution, then the consequence of removing state $s$ is that we delete both strings that are compositionally progressive and strings that are not.

As an example, consider the automata in Fig. 5(c) (largest solution) and Fig. 5(d) (intersection of the context and the largest solution). In Fig. 5(d) there is a non-progressive state $(b, b_1a_2)$ (undefined under input $i_2$). The projection of the language accepted by state $(b, b_1a_2)$ is a proper subset of the language accepted by state $b_1a_2$ in the largest solution; e.g., state $b_1a_2$ of the largest solution accepts the string $v_1u_2\ v_1u_1$ that does not belong to the projection of the language accepted by state $(b, b_1a_2)$ (since the string $i_1v_1u_2o_1\ i_2v_1u_1o_1$ reaches state $(a, b_1a_2)$).

In our example, by deleting state $(b, b_1a_2)$ from the intersection and state $b_1a_2$ from the largest solution, one would delete both

1. the compositionally progressive string $v_1u_2\ v_1u_1$, whose lifting $i_1v_1u_2o_1\ i_2v_1u_1o_1$ drives the intersection to the I-progressive state $(a, b_1a_2)$, and

2. strings that are not compositionally progressive, e.g., $v_1u_1$, whose lifting $i_1v_1u_1o_3$ drives the intersection to the state $(b, b_1a_2)$ that is not I-progressive.

The fact is that in general a state accepts both strings that are compositionally progressive and strings that are not. To avoid it we split such a state into several states is such a way that each state after splitting accepts either only strings that are compositionally progressive or strings that are not. This will hold if the projection of the language accepted by state $as$ coincides with the language accepted by state $s$. The question is how to construct a new automaton with that property and equivalent to the largest solution. We observe that for each string $\alpha$ in the language of the largest solution there is at least one string in the intersection of the context and the solution whose common projection is $\alpha$.

Therefore we merge states with the goal that strings with a common projection drive the intersection into a single state $\hat{s}$ (these states have the same second state component, because the solution automaton is deterministic). As a result, when intersecting the new automaton $S^{FSM}_{split}$ (whose states are merged states, like $\hat{s}$) with the context $A$, the projection of the set of strings that drive the intersection $A \cap S^{FSM}_{split}$ from the initial state to state $a\hat{s}$ is equal to the set of strings that drive $S^{FSM}_{split}$ from the initial state to $\hat{s}$. This is achieved by projecting the intersection of the context and the largest solution, then determinizing without final state minimization of the determinized automaton.

Back to the example, given the string $\alpha = v_1 u_1$ driving the solution to state $b_1 a_2$, the strings $i_1 v_1 u_1 o_3$ and $i_1 v_1 u_1 o_1$ have the same projection $\alpha$ and drive the intersection, respectively, to states $(b, b_1 a_2)$ and $(a, b_1 a_2)$. Therefore, the states $(b, b_1 a_2)$ and $(a, b_1 a_2)$ are merged into a single state $s^\star = \{(a, b_1 a_2), (b, a_1 b_2)\}$ and we get the "restructured" largest solution $S^{FSM}_{split}$, see Fig. 7(c).

The projection of the language accepted by the non-progressive state $s_4 = (b, \{(a, b_1 a_2), (b, b_1 a_2)\})$, where $s_4$ is in the intersection of the context $A$ with the largest solution $S^{FSM}_{split}$, is equal to the language accepted by the state $\{(a, b_1 a_2), (b, b_1 a_2)\}$ in the largest solution $S^{FSM}_{split}$; similarly the projection of the language accepted by the progressive state $s_5 = (a, \{(a, b_1 a_2), (b, b_1 a_2)\})$ is equal to the language accepted by the state $\{(a, b_1 a_2), (b, b_1 a_2)\}$ in the largest solution $S^{FSM}_{split}$.

So, under $v_1 u_1$, $S^{FSM}_{split}$ reaches state $s^\star = \{(a, b_1 a_2), (b, a_1 b_2)\}$, while the context $A$ reaches either state $a$ or state $b$, i.e., the intersection of $A$ and $S^{FSM}_{split}$ reaches either state $s_5 = (a, \{(a, b_1 a_2), (b, b_1 a_2)\})$ or state $s_4 = (b, \{(a, b_1 a_2), (b, b_1 a_2)\})$.

The following procedure tells how to compute $S^{FSM}_{split}$. Notice that in the following, as elsewhere, we blur the distiction between a regular language, say $S^{FSM}$, and the finite automaton that decides it, say $F(S^{FSM})$, and to avoid clumsy notation normally we denote both with $S^{FSM}$.

**Procedure 6.1** *Input: Largest prefix-closed solution $S^{FSM}$ of synchronous inequality $A \bullet X \subseteq C$ and context $A$;*
*Output: $S^{FSM}_{split}$.*

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 6.1** *The automaton $S^{FSM}_{split}$ returned by Proc. 6.1 is such that*

1. *$S^{FSM}_{split}$ is equivalent to the automaton $S^{FSM}$ (i.e., to the automaton deciding the language $S^{FSM}$).*
2. *For all states $a\hat{s}$ of the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$, the projection onto $I_2 \times U \times V \times O_2$ of the language accepted by state $a\hat{s}$ of the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ is equal to the language accepted by state $\hat{s}$ of the automaton $S^{FSM}_{split \uparrow I_1 \times O_1}$ ($\hat{s} \neq dc$ state).*

**Proof.**

1. In order that $S^{FSM}_{split}$ is equivalent to the largest solution $S^{FSM}$ one adds to the automaton obtained by determinization all strings of the largest solution that do not match when composed with the context. Indeed, the automaton obtained by determinizing $R_{\downarrow I_2 \times U \times V \times O_2}$ accepts all strings that can be composed with the context. Each string $\alpha i_2 uvo_2$ of the largest solution such that $\alpha$ can be composed with the context, whereas $\alpha i_2 uvo_2$ cannot, and each extension thereof, take the largest solution to the $dc$ state, so they must be added to $S_{split}$.

2. The statement does not hold for $\hat{s} = dc$ state, because $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ does not have such state, whereas $S^{FSM}_{split \uparrow I_1 \times O_1}$ does. Consider the automaton $S^{FSM}_{split}$, obtained by determinizing $(A_{\uparrow I_2 \times O_2} \cap S_{\uparrow I_1 \times O_1})_{\downarrow I_2 \times U \times V \times O_2}$ and a state $\hat{s}$ of $S^{FSM}_{split}$, where by subset construction $\hat{s}$ is a subset of states of $A_{\uparrow I_2 \times O_2} \cap S_{\uparrow I_1 \times O_1}$, i.e., $\hat{s} = \{a_1 s, \dots, a_k s\}$, $a_i s \in A_{\uparrow I_2 \times O_2} \cap S_{\uparrow I_1 \times O_1}$. The reason why the second item in the pairs of $\hat{s}$ is the same $s$ is that a string $\alpha$ is accepted by a state of this intersection if and only of its projection on $A$ is accepted by a corresponding state of $A$ and its projection on $S$ is accepted by a corresponding state of $S$; since $S$ is deterministic, there is a unique state $s$ to which the projection of $\alpha$ takes $S$ and thus the second item in the pairs of a given $\hat{s}$ is the same. For the given $\hat{s} = \{a_1 s, \dots, a_k s\}$, consider the sets of strings $\{\beta_{a_j s}\}$, $a_j s \in \hat{s}$, where the set $\{\beta_{a_j s}\}$ contains the strings taking the automaton $A_{\uparrow I_2 \times O_2} \cap S_{\uparrow I_1 \times O_1}$ from the initial state to state $a_j s$. By subset construction, the set of strings $\{\alpha_{\hat{s}}\}$ taking the determinization of the automaton $(A_{\uparrow I_2 \times O_2} \cap S_{\uparrow I_1 \times O_1})_{\downarrow I_2 \times U \times V \times O_2}$ from the initial state to state $\hat{s}$ is such that $\{\alpha_{\hat{s}}\} = \bigcap_{a_j s \in \hat{s}} \{\beta_{a_j s}\}_{\downarrow I_2 \times U \times V \times O_2}$, for all $a_j s \in \hat{s}$, i.e., the intersection of the $(I_2 \times U \times V \times O_2)$-projections of the sets $\{\beta_{a_j s}\}$ is exactly equal to the set $\{\alpha_{\hat{s}}\}$. Moreover, each state $as$ such that $\{\beta_{as}\}_{\downarrow I_2 \times U \times V \times O_2}$ intersects $\{\alpha_{\hat{s}}\}$ is in the set $\hat{s}$. Now we prove that the $(I_2 \times U \times V \times O_2)$-projection of the language accepted by state $a\hat{s}$ of the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ is equal to the language accepted by state $\hat{s}$ of the automaton $S^{FSM}_{split}$.

   We show first the $\subseteq$ containment relation. Given state $\hat{s} = \{a_1 s, \dots, a_k s\}$ of the automaton $S^{FSM}_{split}$, consider the state $a\hat{s}$ of the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$. String $\beta$ takes the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ from the initial state to state $a\hat{s}$ only if the $(I_1 \times U \times V \times O_1)$-projection of $\beta$ takes the context $A$ from the initial state to state $a$ and the $(I_2 \times U \times V \times O_2)$-projection of $\beta$ takes the automaton from the initial state to state $\hat{s}$. Thus, the set of $(I_2 \times U \times V \times O_2)$-projections of strings that take the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ from the initial state to state $a\hat{s}$ is a subset of the set $\{\alpha_{\hat{s}}\}$ of strings that take the automaton $S^{FSM}_{split}$ from the initial state to state $\hat{s}$.

   Now we show the $\supseteq$ containment relation. Given a string $\alpha \in \{\alpha_{\hat{s}}\}$, where $\{\alpha_{\hat{s}}\}$ is the set of strings that take the automaton $S^{FSM}_{split}$ from the initial state to state $\hat{s}$, by construction of $S^{FSM}_{split}$ there is a string $\beta$, with $\beta_{\downarrow I_2 \times U \times V \times O_2} = \alpha$, that takes the automaton $A_{\uparrow I_2 \times O_2} \cap S_{\uparrow I_1 \times O_1}$ from the initial state to a state $as$, where $a \in A_{\uparrow I_2 \times O_2}$ and $s \in S_{\uparrow I_1 \times O_1}$. Since $\beta_{\downarrow I_2 \times U \times V \times O_2}$ intersects $\{\alpha_{\hat{s}}\}$ ($\alpha \in \beta_{\downarrow I_2 \times U \times V \times O_2} \cap \{\alpha_{\hat{s}}\}$), $as$ in the set $\hat{s}$, i.e., the string $\beta$ takes the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ from the initial state to state $a\hat{s}$. Therefore, the set of $(I_2 \times U \times V \times O_2)$-projections of strings that take the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ from the initial state to state $a\hat{s}$ contains the set $\{\alpha_{\hat{s}}\}$.

   From the two previous containment relations, we conclude that the $(I_2 \times U \times V \times O_2)$-projection of the language accepted by state $a\hat{s}$ of the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ is equal to the language accepted by state $\hat{s}$ of the automaton $S^{FSM}_{split}$.

□

**Example 6.1** *Referring to Ex. 5.1, consider the largest solution $S$ and the composition $A \cap S_{\uparrow I_1 \times O_1}$, with $S = S^{FSM}$, shown, respectively, in Figs. 5(c) and 5(d). As already noticed, the largest solution $S$ is $U$-progressive, but not compositionally $I$-progressive (there is no transition from $(b, (b1, a2))$ under input $i_2$). Figs. 7(a) and 7(b) show, respectively, the projection of the composition $R = (A \cap S_{\uparrow I_1 \times O_1})_{\downarrow V \times U}$ and its determinization (state $(a, a_1)$ has two transitions under $v_1 u_1$). Adding to the determinized automaton the $dc$ state and related transitions, one obtains $S^{FSM}_{split}$, shown in Fig. 7(c).*

*Notice that states $(a, b_1 a_2)$ and $(b, b_1 a_2)$ are equivalent. A caveat about merging equivalent states: a state that accepts compositionally progressive strings should not be merged with an equivalent state that accepts compositionally non-progressive strings. This explains why in general one cannot obtain the largest compositionally $I$-progressive solution simply by removing states from the largest state-minimized solution.*

Notice that the automaton $S^{FSM}_{split}$ must not be minimized: it is redundant on purpose (and equivalent to $S^{FSM}$) to serve as starting point to the following procedure that deletes states to obtain the largest compositionally progressive solution.

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

**Procedure 6.2** *Input: Largest prefix-closed solution $S^{FSM}_{split}$ of synchronous inequality $A \bullet X \subseteq C$ 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 states $B$ of $R^i$ that are not $I$-progressive, i.e., $B = \{sr\}$, $sr \in R^i, s \in A_{\uparrow I_2 \times O_2}, r \in S^i_{\uparrow I_1 \times O_1}$ and $sr$ is not $I$-progressive.

   (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 $sr \in B$ ($r \in S^i$ if and only if $r \in S^i_{\uparrow I_1 \times O_1}$).

   (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 $sr \in B$.

   ii. Increment $i$ by 1 and go to 3.

**Theorem 6.2** *Proc. 6.2 returns the largest compositionally $I$-progressive (prefix-closed) solution.*

**Proof.** We define inductively *$k$-non-cprogressive* states, $k > 0$, as follows. A state $\hat{s}$ of the automaton $S^{FSM}_{split}$ is called 1-non-cprogressive if there is a state $a \in A_{\uparrow I_2 \times O_2}$ such that $a\hat{s}$ is a state of $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ and $a\hat{s}$ is not $I$-progressive. A state $\hat{s}$ is called ($k$+1)-non-cprogressive if it is $k$-non-cprogressive or there exists a state $a\hat{s}$ of the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ and an input $\tilde{i}_1 \tilde{i}_2 \in I$ such that every transition from $a\hat{s}$ of the type $\tilde{i}_1 \tilde{i}_2 u v o_1 o_2$ takes the automaton to a state whose second component is a $k$-non-cprogressive

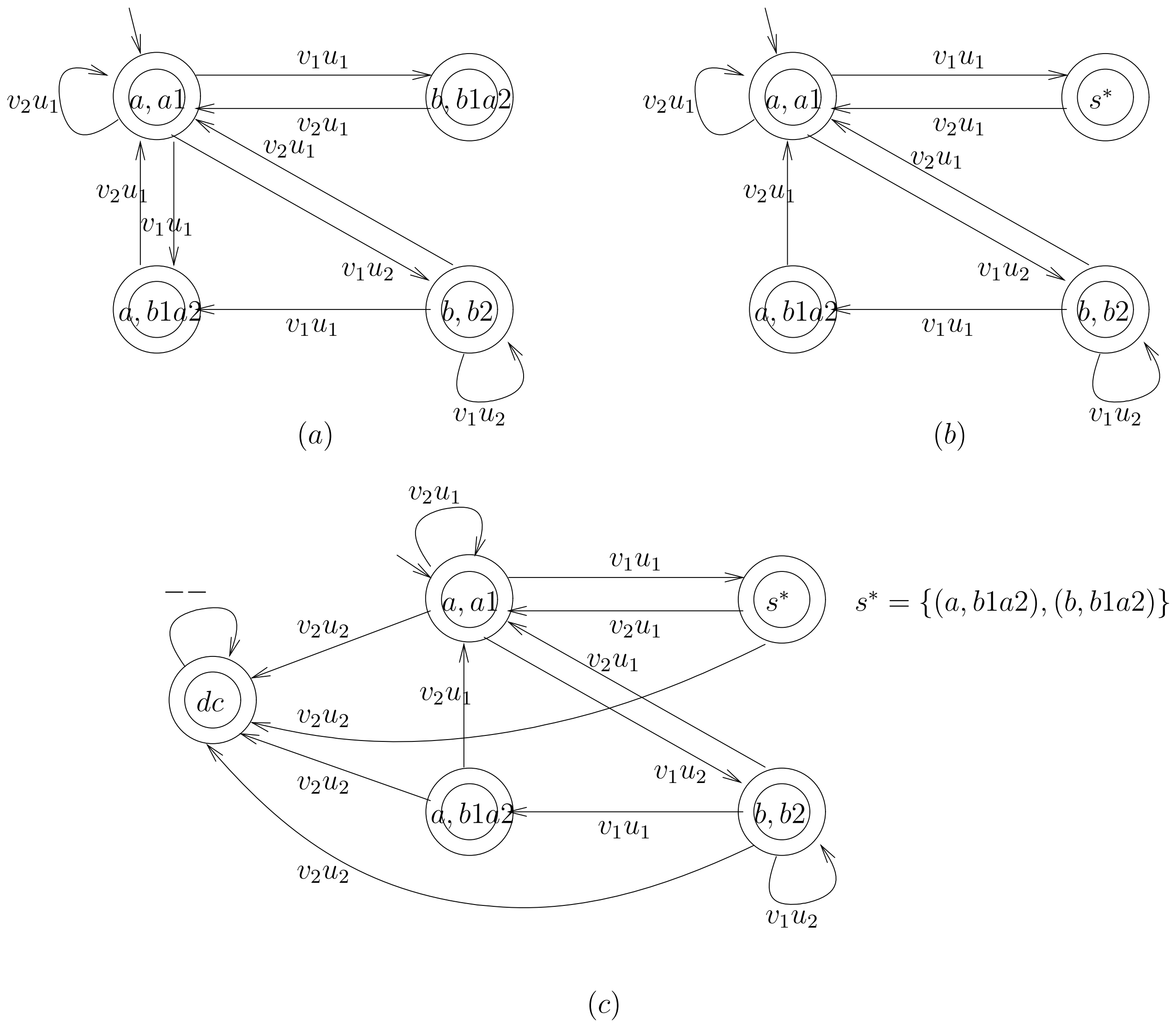

Figure 7: Illustration of Example 6.1. (a) $(A \cap S_{\uparrow I_1 \times O_1})_{\downarrow V \times U}$; (b) determinization of $(A \cap S_{\uparrow I_1 \times O_1})_{\downarrow V \times U}$; (c) $S^{FSM}_{split}$.

state. A state $\hat{s}$ is called *non-cprogressive* if it is $k$-non-cprogressive for an appropriate $k$; otherwise, $\hat{s}$ is called *cprogressive*.

Since Proc. 6.2 deletes from $S^{FSM}_{split}$ only non-cprogressive states, to prove the thesis it is sufficient to show that a compositionally progressive solution has no string that takes $S^{FSM}_{split}$ to a non-cprogressive state. The proof goes by induction on index $k$.

**Base case**

Let $\hat{s}$ be a 1-non-cprogressive state of $S^{FSM}_{split}$. By definition of 1-non-cprogressive, there exists a state $a\hat{s}$ of the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ that is not $I$-progressive. Moreover, by the characterization of $S^{FSM}_{split}$ established by Th. 6.1, for each string $\alpha$ that takes the automaton $S^{FSM}_{split}$ from the initial state to state $\hat{s}$ there exists a string $\beta$, with $\beta_{\downarrow I_2 \times U \times V \times O_2} = \alpha$, that takes the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ from the initial state to state $a\hat{s}$. Since state $a\hat{s}$ is not $I$-progressive, a string $\alpha$ taking the automaton $S^{FSM}_{split}$ from the initial state to a 1-non-cprogressive state is not in a compositionally progressive solution. That proves the base case by contradiction.

**Inductive step**

The inductive hypothesis is that a compositionally progressive solution has no string that takes the automaton $S^{FSM}_{split}$ from the initial state to a $k$-non-cprogressive state, $0 < k < m$. Assume by contradiction that a compositionally progressive solution has a string $\alpha$ that takes the automaton $S^{FSM}_{split}$ from the initial state to an $m$-non-cprogressive state, $\hat{s}$. Then, again by the characterization of $S^{FSM}_{split}$ by Th. 6.1, for each state $a\hat{s}$ of the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$, there exists a string $\beta$, with $\beta_{\downarrow I_2 \times U \times V \times O_2} = \alpha$, which takes the automaton $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{split \uparrow I_1 \times O_1}$ from the initial state to the state $a\hat{s}$; moreover, by definition of $m$-non-cprogressive state $\hat{s}$, (since it is not $(m-1)$-non-cprogressive by induction hypothesis) there is an input $\tilde{i}_1\tilde{i}_2 \in I$, such that every transition from $a\hat{s}$ of the type $\tilde{i}_1\tilde{i}_2 uvo_1o_2$ takes the automaton to a state whose second component is a $(m-1)$-non-cprogressive state.

Since by induction hypothesis state $\hat{s}$ is not 1-non-cprogressive, for the given $\tilde{i}_1\tilde{i}_2$ there is a quadruple $uvo_1o_2$ such that the given string $\alpha$ can be extended, from state $\hat{s}$, by the $(I_2 \times U \times V \times O_2)$-projection of $\tilde{i}_1\tilde{i}_2 uvo_1o_2$; by the previous observation [that every transition from $a\hat{s}$ of the type $\tilde{i}_1\tilde{i}_2 uvo_1o_2$ takes the composition automaton to a state whose second component is a $(m-1)$-non-cprogressive state] also the 6-tuple $\tilde{i}_1\tilde{i}_2 uvo_1o_2$ takes the composition automaton from state $a\hat{s}$ to a state whose second component is a $(m-1)$-non-cprogressive state. Therefore, the compositionally progressive solution has a string that takes the automaton $S^{FSM}_{split}$ from the initial state to a $(m-1)$-non-cprogressive state, which contradicts the inductive hypothesis. $\Box$

**Example 6.2** *Fig. 8(a) shows $R^1$ computed by step 2. of Proc 6.2. Since state $(b, \{(a, b_1a_2), (b, b_1a_2)\})$ is not defined under $i_2$, the language of automaton $R^1$ is not $I$-progressive. The set of incompletely specified states is $B = \{(b, \{(a, b_1a_2), (b, b_1a_2)\})\}$ (no transitions under $i_2$). Since $B$ contains only one state, we remove from $S^1$ the state $s^\star = \{(a, b_1a_2), (b, b_1a_2)\}$ and obtain $S^2$ drawn in Fig. 8(b). The same witness $s = b$ (from $(b, \{(a, b_1a_2), (b, b_1a_2)\}) \in B$) testifies that the states $s_4 = (b, \{(a, b_1a_2), (b, b_1a_2)\})$ and $s_5 = (a, \{(a, b_1a_2), (b, b_1a_2)\})$ must be removed from $R^1$, yielding $R^2$, as shown in Fig. 8(c). Given that $R^2$ is $I$-progressive, it results that $S^2$ is the largest compositionally progressive solution.*

**Example 6.3** *Figs. 9 and 11 show the main steps to compute the largest compositionally progressive solution for the FSM equation given in Figs. 9(a)-(b), whose largest solution is in Fig. 9(c). Figs. 10(a)-(e) show some detailed intermediate steps in the computations of Fig. 9.*

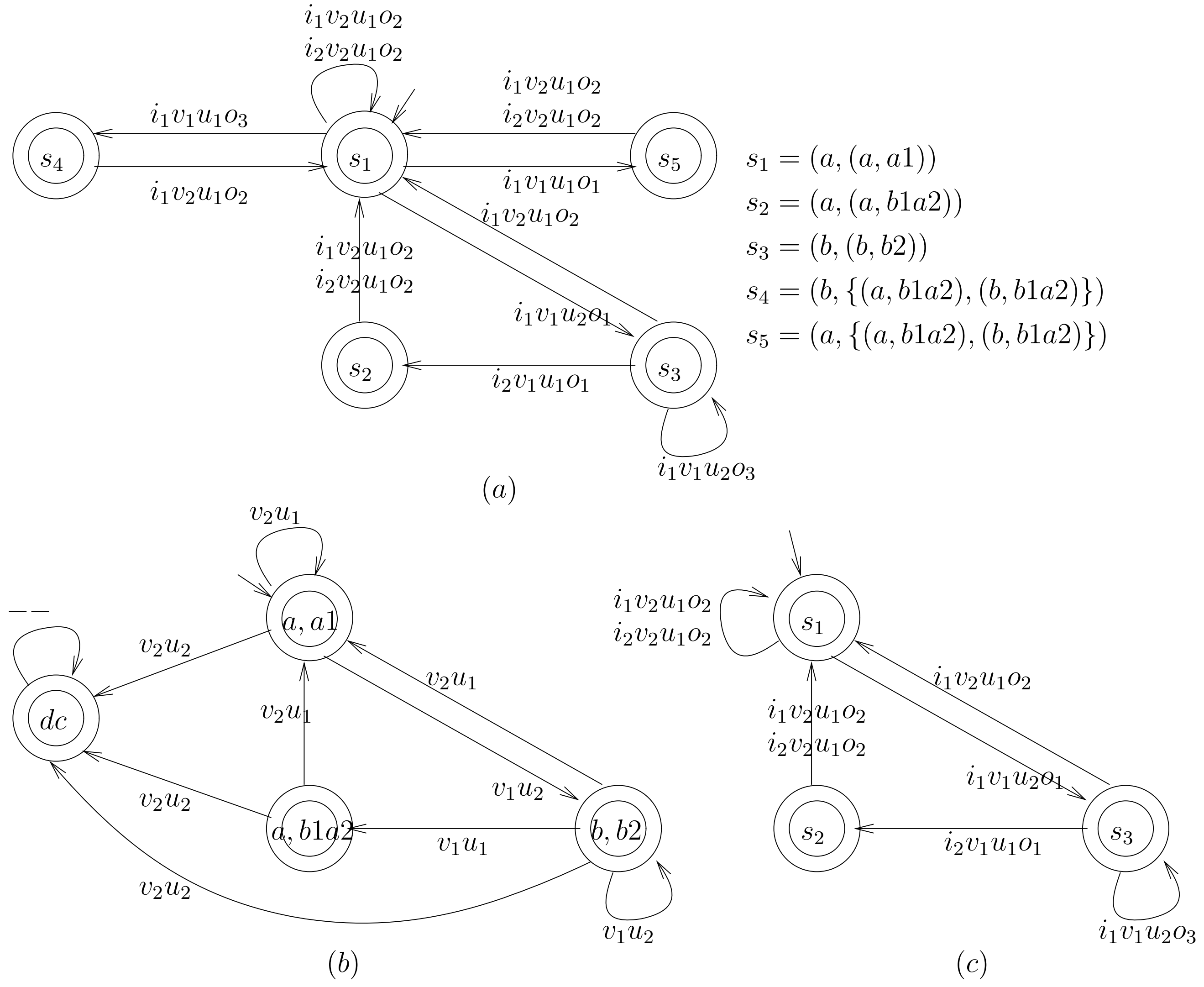

Figure 8: Illustration of Example 6.2. (a) $R^1 = A \cap S^1_{\uparrow I_1 \times O_1}$, where $S^1 = S^{FSM}_{split}$; (b) $S^2$; (c) $R^2$.

*The results of the main steps of Proc. 5.1 are presented in Figs. 9(d)-(l), whereas the main steps of Procs. 6.1 and 6.2 are given in Figs. 11(a)-(g). In the latter computation, the set of incompletely specified states of $R^1$ is given by $B = \{(b, \{a1, b1\})\}$ (state $(b, \{a1, b1\})$ is not specified under $i_2$). Then $S^2$ is obtained by deleting state $\{a1, b1\}$ from $S^1$, and $R^2$ is obtained by deleting states $(a, \{a1, b1\})$ and $(b, \{a1, b1\})$ from $R^1$. Since $R^2$ is $I$-progressive, it follows that $S^2 = cProg(S^{FSM})$.*

**Example 6.4** *Applying either the string removal procedure or the state splitting and removal procedure to the equation of Example 3.1, one obtains the largest compositionally progressive solution anticipated in Fig. 2(c).*

If $|A|$ ($|S|$) is the number of states of context $A$ (largest solution $S$), the worst-case computational complexity of Proc. 6.1 is at least $O(2^{|A||S|})$ (a sharper bound is to be determined), and the worst-case computational complexity of Proc. 6.2 is $O(|A|.2^{|A||S|})$.

## 7 Conclusions and Future Work

We investigated compositionally progressive solutions of synchronous FSM inequalities and 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.

They are solved by reducing them to inequalities over languages associated to the FSMs and studying compositionally progressive solutions of language inequalities. We showed that if there is a compositionally progressive language solution, then there is the largest compositionally progressive language solution. If the largest language solution is compositionally progressive, then it is the largest compositionally progressive language solution. If the composition of the context with the largest compositionally progressive solution of the inequality is equivalent to the specification then it is the largest compositionally progressive solution of the equation.

Otherwise, input-output strings that violate the property must be deleted from those accepted by the largest compositionally progressive solution, until what is left is compositionally progressive or it is empty (there is no such solution).

We explored two options: either delete all compositionally non-progressive strings from the largest solution or split states of the largest solution into equivalent states in such a way that each state of the modified automaton $S_{split}$ is reachable from the initial state either only by compositionally progressive strings or only by compositionally non-progressive strings. A sub-automaton of $S_{split}$ restricted to the states reachable by compositionally progressive strings yields the largest compositionally progressive solution, if it exists. At the end an FSM is obtained whose language is the largest compositionally progressive solution computed by either algorithm. Since there may be more than one FSM corresponding to that language we say that it is a largest compositionally progressive FSM solution.

It is noticeable that it may be the case that a largest compositionally progressive FSM solution is not a submachine of any largest FSM solution, even though experiments showed this to be a rare occurrence in practice. Instead a largest FSM solution that is complete and a largest FSM solution that is Moore are a submachine of any largest FSM solution.

The theory outlined in this paper can be extended to more general formulations of the problem, e.g. when the FSMs are not complete or are not observable. For instance, a more general problem would be, given the specification $M_C$ whose language is not $I$-progressive, find a solution $M_B$ that is:

- at least as progressive as $M_C$, i.e., the automaton of $L(M_A)_{\uparrow I_2 \times O_2} \cap L(M_B)_{\uparrow I_1 \times O_1}$ is free to do whatever it wants when the automaton of $L(M_C)$ is undefined, or,

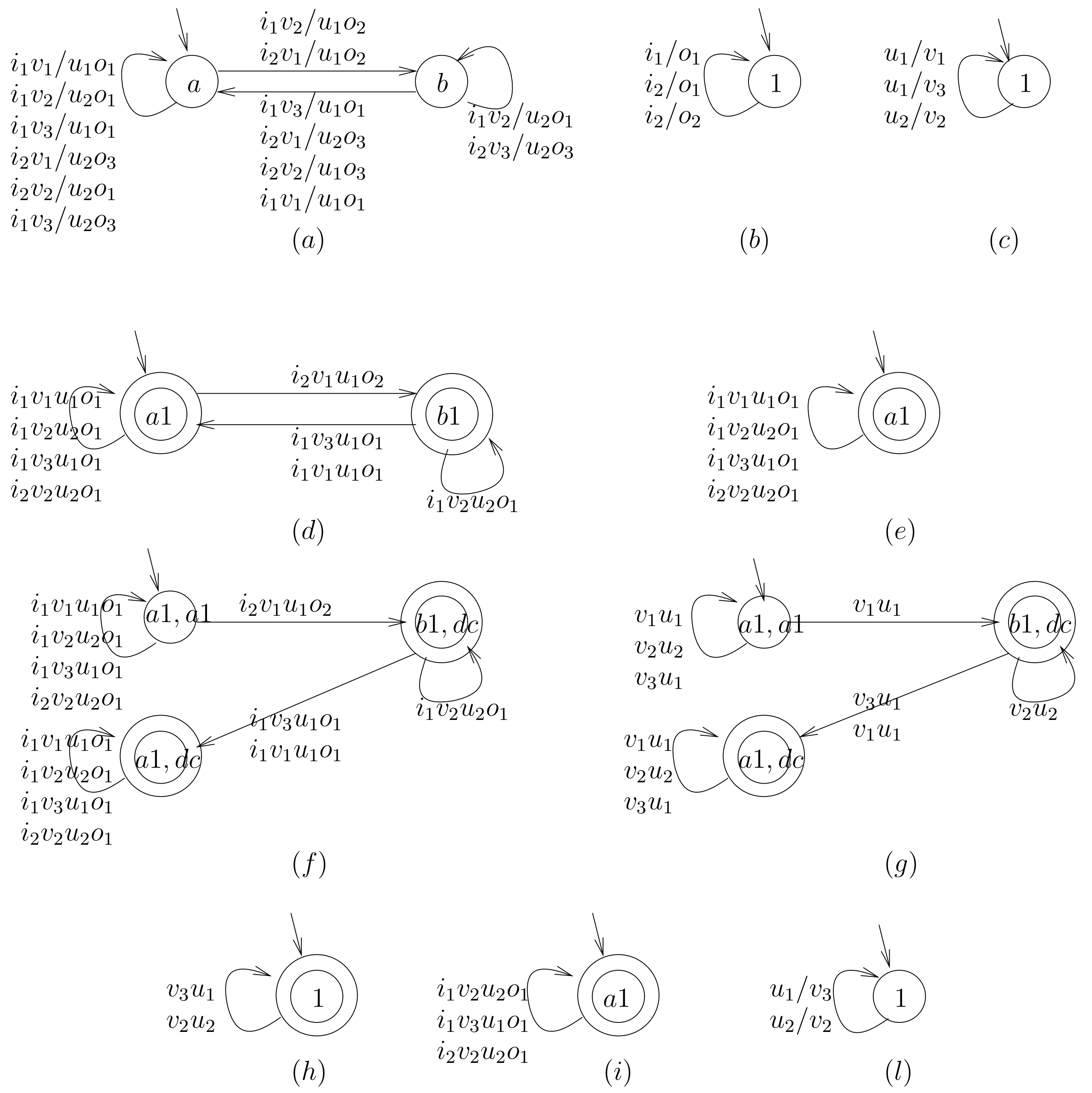

Figure 9: Illustration of Example 6.3. (a) FSM $M_A$; (b) FSM $M_C$; (c) Largest solution $M_S$; (d) $R^1 = A \cap S^1_{\uparrow I_1 \times O_1}$, with $S^1 = S^{FSM}$; (e) $Prog(R^1)$; (f) $R^1 \setminus Prog(R^1)$; (g) $B^1 = (R^1 \setminus Prog(R^1))_{\downarrow V \times U}$; (h) $T^1 = S^1 \setminus B^1$; (i) $R^2 = A \cap S^2_{\uparrow I_1 \times O_1}$, with $S^2 = T^{1\ FSM}$; (l) Largest compositionally progressive FSM solution $M_{S^2}$.

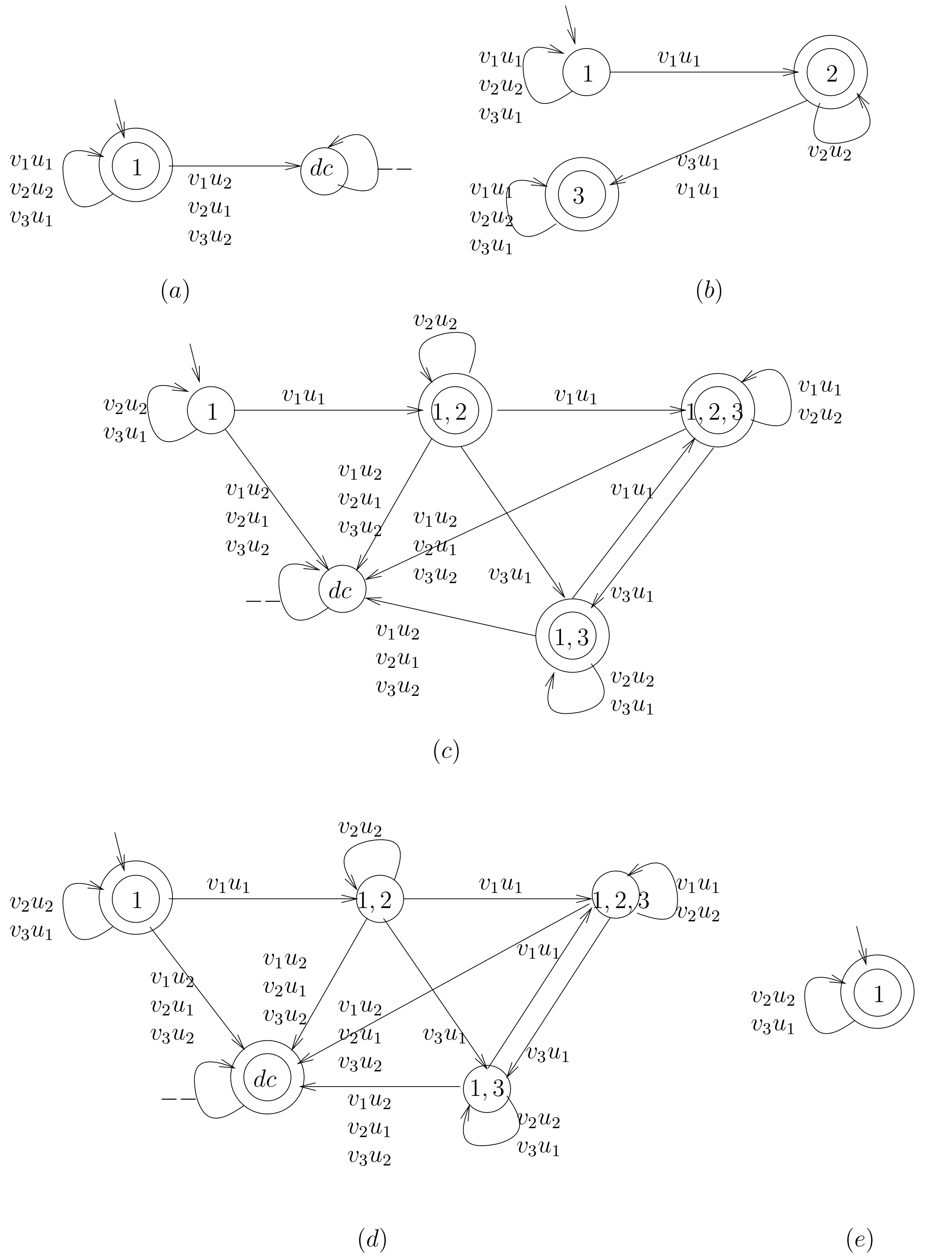

Figure 10: Illustration of Example 6.3. (a) $S^1$; (b) $B^1$; (c) Determinization of $B^1$; (d) $\overline{B^1}$; (e) $T^1 = S^1 \setminus B^1$.

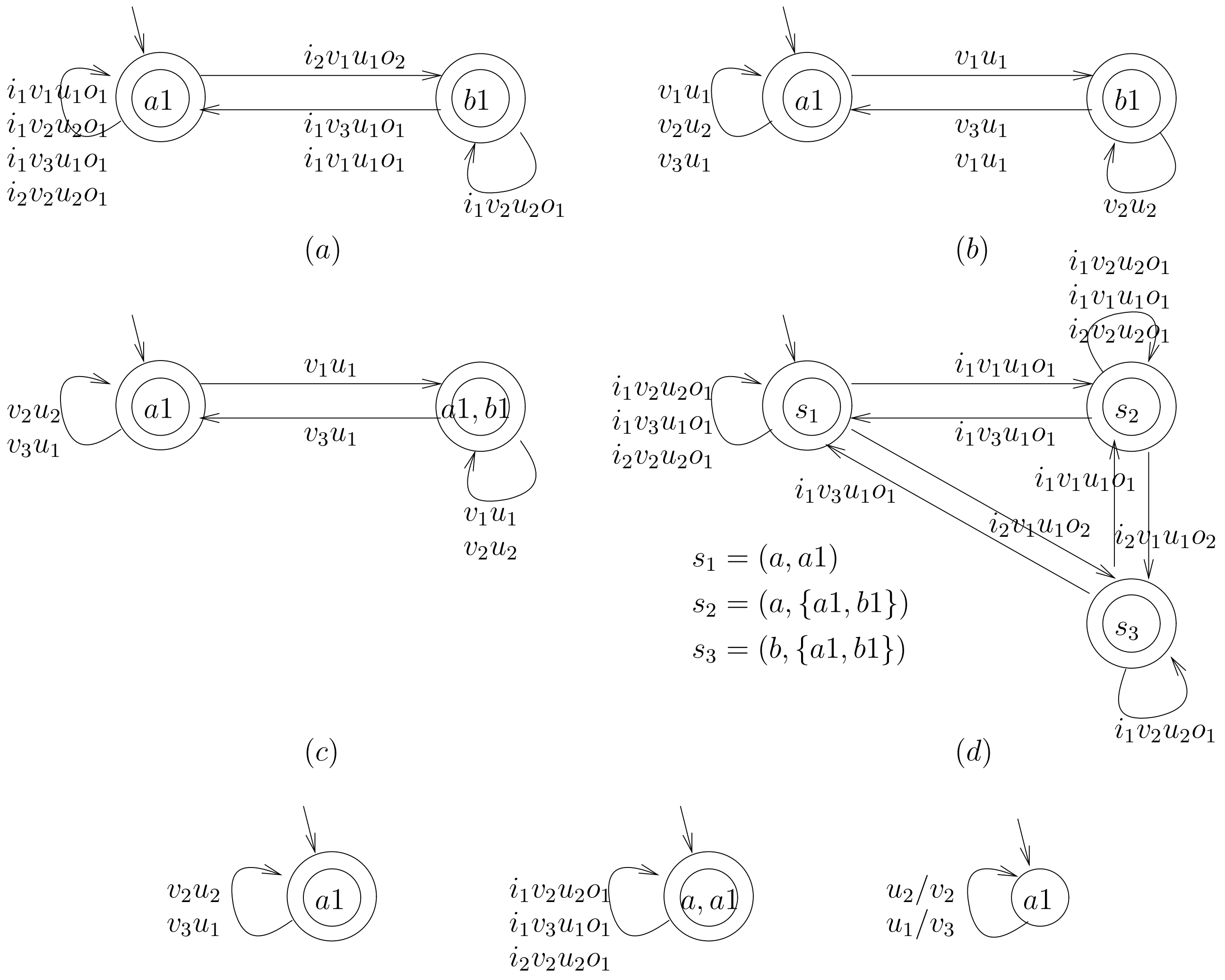

Figure 11: Illustration of Example 6.3. (a) $A \cap S_{\uparrow I_1 \times O_1}$, where $S = S^{FSM}$; (b) $(A \cap S_{\uparrow I_1 \times O_1})_{\downarrow U \times V}$; (c) Determinization of $(A \cap S_{\uparrow I_1 \times O_1})_{\downarrow U \times V} = S^{FSM}_{split}$; (d) $R^1 = A \cap S^1_{\uparrow I_1 \times O_1}$, where $S^1 = S^{FSM}_{split}$; (e) $S^2$ obtained by deleting state $\{a1, b1\}$ from $S^1$; (f) $R^2$ obtained by deleting states $(a, \{a1, b1\})$ and $(b, \{a1, b1\})$ from $R^1$; (g) Largest compositionally progressive FSM solution $M_{S^2}$.

- exactly as progressive as $M_C$, i.e., the automaton of $L(M_A)_{\uparrow I_2 \times O_2} \cap L(M_B)_{\uparrow I_1 \times O_1}$ blocks exactly the inputs blocked by the automaton of $L(M_C)$.

Notice that partial specification, i.e., a state is undefined ('blocked') under input $i$[3], could be modeled explicitly as follows:

1. Add a self-loop on input $i$ with a designated output ERROR (the systems informs of an invalid input, but it continues to function), or,

2. Add a transition on input $i$ with a designated output ERROR to an ERROR state that outputs ERROR on every input (the systems fails under an invalid input and cannot resume its normal operation).

Finally, some of the alphabets on which $M_A$, $M_B$ and $M_C$ are defined may be empty and so the lifting or projection operations will be performed only with respect to the non-empty alphabets. Moreover, some internal signals may be observable, i.e., $U$ and/or $V$ may be also external outputs. The results of this paper can be extended also to these more general topologies.

A further variant of the problem would be to enforce *output-progressiveness*, meaning that no output that can be produced by the specification should be blocked.

Future work includes studying restricted classes of equations for which we can guarantee a complete characterization of compositionally progressive solutions, i.e., we can compute $S_{CP}$ that contains (all and) only complete solutions that are compositionally progressive, as done for the series and controller's topologies by M. Vetrova in her dissertation [12].

## 8 Acknowledgments

The first author was partly supported by the project 06-08-89500 of the joint scientific program RFBR-NSC. The first three authors gratefully acknowledge the support of a NATO travel grant (NATO Linkage Grant No. PST. CLG. 979698). The fourth author gratefully acknowledges the support of NSERC (Grant OGP0194381).

## References

[1] G. Barrett and S. Lafortune. Bisimulation, the supervisory control problem and strong model matching for finite state machines. *Discrete Event Dynamic Systems: Theory & Applications*, 8(4):377–429, December 1998.

[2] S. Buffalov, K. El-Fakih, N. Yevtushenko, and G. v. Bochmann. Progressive solutions to a parallel automata equation. In *Proceedings of the IFIP 23rd International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2003)*, volume 2767 of *LNCS*, pages 367–382. Springer Verlag, September 2003.

[3] C. C. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Kluwer Academic Publishers, 1999.

[3] This is completely different from non-determinism, where there is a transition on input $i$ with all outputs to a 'don't care' state that has all outputs on each input.

[4] K. El-Fakih, S. Buffalov, N. Yevtushenko, and G. v. Bochmann. Progressive solutions to a parallel automata equation. *Theoretical Computer Science*, 362:17–32, 2006.

[5] L.E. Holloway, B.H. Grogh, and A. Giua. A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications*, Vol. 7(No. 2):151–190, April 1997.

[6] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 2001.

[7] R. Kumar, S. Nelvagal, and S.I. Marcus. A discrete event systems approach for protocol conversion. *Discrete Event Dynamic Systems: Theory & Applications*, 7(3):295–315, June 1997.

[8] A. Mishchenko, R. Brayton, J.-H. Jiang, T. Villa, and N. Yevtushenko. Efficient solution of language equations using partitioned representations. In *The Proceedings of the Design, Automation and Test in Europe Conference*, volume 01, pages 418–423, March 2005.

[9] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, Vol. 77(No. 1):81–98, January 1989.

[10] P. H. Starke. *Abstract Automata*. North-Holland Pub. Co.; American Elsevier Pub. Co., 1972.

[11] G. v. Bochmann. Submodule construction and supervisory control: a generalization. In B.W. Watson and D. Wood, editors, *Proceedings of the 6th International Conference on Implementation and Application of Automata (CIAA '01)*, volume 2494 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2002.

[12] Maria Vetrova. *Designing and Testing FSM compensators*. PhD thesis, Tomsk State University, Russia, 2004. (in Russian).

[13] T. Villa, N. Yevtushenko, and S. Zharikova. Characterization of progressive solutions of a synchronous FSM equation. In *Vestnik, 278, Series Physics*, pages 129–133, sep 2003. (In Russian).

[14] 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, http://www.parades.rm.cnr.it/ villa/articoli/ps/tomsk2002.ps.

[15] N. Yevtushenko, T. Villa, R. Brayton, A. Petrenko, and A. Sangiovanni-Vincentelli. Sequential synthesis by language equation solving. Technical report, Tech. Rep. No. UCB/ERL M03/9, Berkeley, CA, April 2003, http://www.parades.rm.cnr.it/ villa/articoli/ps/TR-UCB_ERL-M03_9.ps.

# Appendix

## A Proof that Proc. 5.1 Terminates

The following proposition has been proved in [15].

**Proposition A.1** *The following relations hold.*
*(a) Given alphabets $X$ and $Y$, and a language $L$ over alphabet $X$, then $(L_{\uparrow Y})_{\downarrow X} = L$.*
*(b) Given alphabets $X$ and $Y$, and a language $L$ over alphabet $X \times Y$, then $(L_{\downarrow X})_{\uparrow Y} \supseteq L$.*

**Proposition A.2** *The following distributive laws for $\uparrow$ and $\downarrow$ hold.*
*(a) Let $L_1, L_2$ be languages over alphabet $U$. Then $\uparrow$ commutes with $\cup$*

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

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

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

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

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

*(d) 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 $\cap$*

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

**Theorem A.1** *Proc. 5.1 terminates.*

**Proof.** We show by induction that the language $R^i = A_{\uparrow I_2 \times O_2} \cap S^i_{\uparrow I_1 \times O_1}$ is recognized by the DFA $F^i = \langle S^i, I_1 \times I_2 \times U \times V \times O_1 \times O_2, \Delta^i, r^i, Q^i \rangle$, such that each accepting state $q^i_j \in Q^i,\ j \in J^i = \{1, \ldots, |Q^i|\}$, recognizes the language $L(q^i_j)$ =

$$\bigcup_{\substack{H \in \mathcal{H}^i_j \subseteq 2^{J^1} \\ K \in \mathcal{K}^i_j \subseteq 2^{J^1}}} [\ L(q^1_{l^i_j}) \cap \bigcap_{h \in H \subseteq J^1} (L(q^1_h)_{\downarrow I_2 \times U \times V \times O_2})_{\uparrow I_1 \times O_1} \cap \bigcap_{k \in K \subseteq J^1} (\overline{L(q^1_k)_{\downarrow I_2 \times U \times V \times O_2}})_{\uparrow I_1 \times O_1}\ ], \tag{3}$$

where $l^i_j \in J^1, \mathcal{H}^i_j \subseteq 2^{J^1}$, $\mathcal{K}^i_j \subseteq 2^{J^1}$, and $q^1_{l^i_j}, q^1_h, q^1_k \in Q^1$. Moreover, $H, K \neq \emptyset$, since we consider only non-empty subsets of states of $R^{i-1}$; without repeating it each time, we will consider only non-empty subesets $H$ and $K$ also in related formulas of the induction step.

Therefore the language $R^i$ can be represented as a finite combination (through unions and intersections) of the languages and/or complements (under appropriate liftings and projections) of the states of the DFA accepting $R^i$. Therefore the number of such languages is finite. Moreover, $S^i \subset S^{i-1}$ and, correspondingly, $R^i \subset R^{i-1}$, i.e., the sequence of languages $\{R^i\}$ must terminate either with the empty language or with a progressive language.

**Induction basis** The basis of the induction is shown by the previous observation on the expressions that represent the languages recognized by the states of the DFAs accepting $R^1$ and $R^2$.

**Induction step** Suppose that the induction assumption (specified by Eq. 3)) holds for $R^{i-1}$ and show that it holds also for $R^i$.

The following derivation expresses $R^i$ as a function of $R^{i-1}$:

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

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

By the induction hypothesis, in the equation

$$R^i = R^{i-1} \cap \overline{B^{i-1}}_{\uparrow I_1 \times O_1}$$

language $R^{i-1}$ is recognized by the automaton $F^{i-1}$ such that each state of $F^{i-1}$ accepts a language, which can be expressed as a composition of languages recognized by states of $F_1$ according to Eq. 3. Our goal is to show that also language $R_i$ has the same property.

Let us examine the constructive steps leading from language $R^{i-1}$ to language $R^i$.

- Language $R^{i-1} \setminus Prog(R^{i-1})$. It is recognized by the DFA $F^{i-1}_{NP} = \langle S^{i-1}, I_1 \times I_2 \times U \times V \times O_1 \times O_2, \Delta^{i-1}, r^{i-1}, Q^{i-1}_{NP} \rangle$, where $Q^{i-1}_{NP}$ is the subset of non-progressive states of $Q^{i-1}$. Each accepting state $q^{i-1} \in Q^{i-1}_{NP}$ recognizes the language $L(q^{i-1})$, which by induction hypothesis can be expressed in the form of Eq. 3:

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

  where $l^{i-1}_j \in J^1, \mathcal{H}^{i-1}_j \subseteq 2^{J^1}, \mathcal{K}^i_j \subseteq 2^{J^1}$, and $q^1_{l^{i-1}_j}, q^1_h, q^1_k \in Q^1$.

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

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

  where $l^{i-1}_j \in J^1, \mathcal{H}^{i-1}_j \subseteq 2^{J^1}$, $\mathcal{K}^i_j \subseteq 2^{J^1}$, and $q^1_{l^{i-1}_j}, q^1_h, q^1_k \in Q^1$. We used the fact that $(M_1 \cup M_2)_{\downarrow U} = M_{1\,\downarrow U} \cup M_{2\,\downarrow U}$ and $(M_1 \cap M_2)_{\downarrow U} = M_{1\,\downarrow U} \cap M_{2\,\downarrow U}$ if $M_2 = (M_{2\,\downarrow U})_{\uparrow I}$ by Prop. A.2.

- Language $R^i = R^{i-1} \cap (\overline{(R^{i-1} \setminus Prog(R^{i-1}))_{\downarrow I_2 \times U \times V \times O_2}})_{\uparrow I_1 \times O_1} = R^{i-1} \cap \overline{B^{i-1}}_{\uparrow I_1 \times O_1}$. To complement $B^{i-1}$, the NDFA of $B^{i-1}$ must be determinized yielding a DFA $F^{i-1}_{\hat{B}^{i-1}} = \langle \hat{S}^1, I_2 \times U \times$

$V \times O_1, \hat{\Delta}^{i-1}, \hat{r}^{i-1}, \hat{Q}^{i-1}\rangle$ whose states are subsets of states of $B^{i-1}$; 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 $\hat{q}_j^{i-1} \in \hat{Q}^{i-1},\ j \in \hat{J}^{i-1} = \{1, \ldots, |\hat{Q}^{i-1}|\}$ is such that $\hat{q}_j^{i-1} = \{q_k^{i-1} \mid k \in K_j^{i-1} \subseteq J^{i-1}\}$ and $L(\hat{q}_j^{i-1}) = \bigcap_{k \in K_j^{i-1} \subseteq J^{i-1}} L(q_k^{i-1})_{\downarrow I_2 \times U \times V \times O_2}$. Therefore, by intersecting the languages of the states in the related subset, the language of each state can be expressed in the form

$$\bigcap [ \bigcup_{\substack{H \in \mathcal{H}_j^i \subseteq 2^{J^1} \\ K \in \mathcal{K}_j^i \subseteq 2^{J^1}}} [ L(q^1_{l_j^{i-1}})_{\downarrow I_2 \times U \times V \times O_2} \cap \bigcap_{h \in H \subseteq J^1} L(q_h^1)_{\downarrow I_2 \times U \times V \times O_2} \cap \bigcap_{k \in K \subseteq J^1} \overline{L(q_k^1)_{\downarrow I_2 \times U \times V \times O_2}} ]],$$

where $l_j^{i-1} \in J^1, \mathcal{H}_j^{i-1} \subseteq 2^{J^1}, \mathcal{K}_j^i \subseteq 2^{J^1}$, and $q^1_{l_j^{i-1}}, q_h^1, q_k^1 \in Q^1$; by algebraic manipulation the latter expression can be rewritten as

$$\bigcup_{\substack{H \in \mathcal{H}_j^i \subseteq 2^{J^1} \\ K \in \mathcal{K}_j^i \subseteq 2^{J^1}}} [ \bigcap_{h \in H \subseteq J^1} L(q_h^1)_{\downarrow I_2 \times U \times V \times O_2} \cap \bigcap_{k \in K \subseteq J^1} \overline{L(q_k^1)_{\downarrow I_2 \times U \times V \times O_2}} ]. \qquad (4)$$

Also 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: also $L(\{dc\})$ after some algebra can be expressed in the form of Eq. 4. The DFA of $\overline{B^{i-1}}$ is obtained by switching accepting and non-accepting states of the determinization of $B^{i-1}$. Finally the language $R^i$ is recognized by the DFA $F^i = \langle S^i, I_1 \times I_2 \times U \times V \times O_1 \times O_2, \Delta^i, r^i, Q^i\rangle$, obtained by intersection of the automata of $R^{i-1}$ and $\overline{B^{i-1}}_{\uparrow I_1 \times O_1}$. So each accepting state $q_j^i \in Q^i,\ j \in J^i = \{1, \ldots, |Q^i|\}$, recognizes the language $L(q_j^i)$ expressed in the form of Eq. 3:

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

where $l_j^i \in J^1, \mathcal{H}_j^i \subseteq 2^{J^1}, \mathcal{K}_j^i \subseteq 2^{J^1}$, and $q^1_{l_j^i}, q_h^1, q_k^1 \in Q^1$. 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. A.2.

□