SEQUENTIAL SYNTHESIS BY LANGUAGE EQUATION SOLVING

by

Nina Yevtushenko, Tiziano Villa, Robert K. Brayton,
Alex Petrenko and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M03/9

11 April 2003

SEQUENTIAL SYNTHESIS BY LANGUAGE EQUATION SOLVING

by

Nina Yevtushenko, Tiziano Villa, Robert K. Brayton,
Alex Petrenko and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M03/9

11 April 2003

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Sequential Synthesis by Language Equation Solving

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

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

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

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

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

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

April 11, 2003

Abstract

Consider the problem of designing a component that combined with a known part of a system, called the context, conforms to a given overall specification. This question arises in several applications ranging from logic synthesis to the design of discrete controllers.

We cast the problem as solving abstract equations over languages and study the most general solutions under the synchronous and parallel composition operators. We also specialize such language equations to languages associated with important classes of automata used for modeling systems, e.g., regular languages as counterparts of finite automata, FSM languages as counterparts of FSMs. Thus we can operate algorithmically on those languages through their automata and study how to solve effectively their language equations. We investigate the maximal subsets of solutions closed with respect to various language properties. In particular, we investigate classes of the largest compositional solutions (defined by properties exhibited by the composition of the solution and of the context). We provide the first algorithm to compute the largest compositionally progressive solution of synchronous equations.

This approach unifies in a seamless frame previously reported techniques.

As an application we solve the classical problem of synthesizing a converter between a mismatched pair of protocols, using their specifications, as well as those of the channel and of the required service.

1 Introduction

An important step in the design of complex systems is the decomposition of a system into a number of separate components which interact in some well-defined way. A typical question is how to design a component that when combined with a known part of the system, called the context, satisfies a given overall specification. This question arises in several applications ranging from logic synthesis to the design of discrete controllers. Some common network topologies are shown in Figure 1. To formally solve such problems, the following questions need to be addressed:

- How to model the system, its components and the specification
- How is the interaction between components defined
- When does a system behavior satisfy its specification

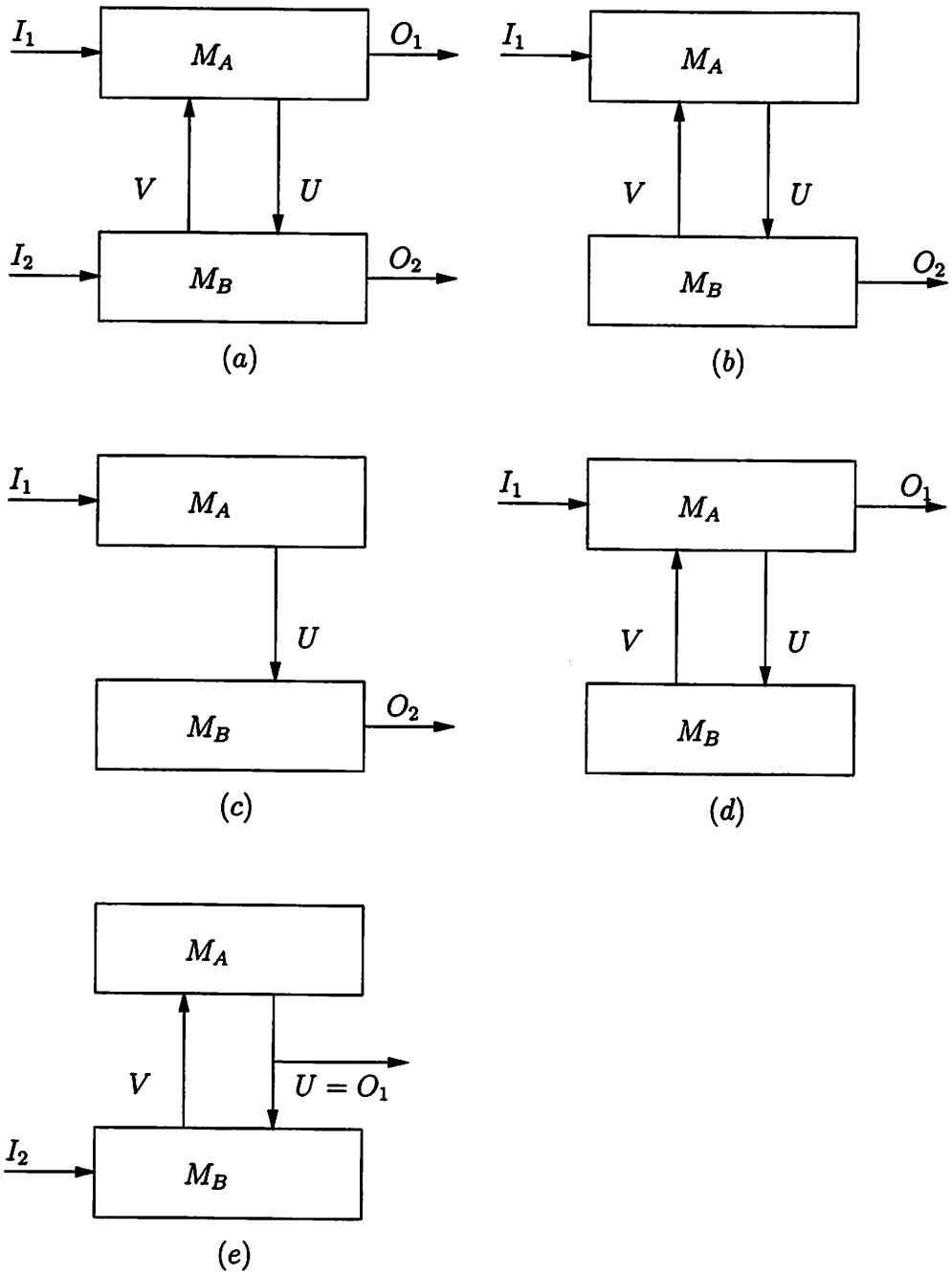

Figure 1: Patterns of composition. (a) general topology; (b) 2-way cascade (or 2-way series) topology; (c) 1-way cascade (or 1-way series) topology; (d) rectification topology; (e) controller's (or supervisory control) topology.

For the first issue, different types of mathematical machines can be used to model the components of a system: finite automata (FA), finite state machines (FSMs), Petri Nets (PNs), ω-automata (ω-FA) are used most commonly. Given a decision on the first issue, related choices must be made for the other two. For instance, if FSMs are used to model the system, operators to compose FSMs must be introduced together with the notion of an FSM conforming to another FSM. For the last issue popular choices are language containment or simulation of one FSM by the other. For FSM composition, various forms have been described in the literature. For example, one can define an equation over FSMs of the type $M_A \odot M_X \approx M_C$, where M_A models the context, M_C models the specification, M_X is unknown, $\odot$ stands for a composition operator and $\approx$ for a conforming relation (e.g., $\subseteq$, language containment). For any given model of mathematical machines, appropriate equations can be set up and their solutions investigated. More complex equations or systems of equations can be formulated depending on the topology of the system's components.

A useful observation is that a certain class of languages is associated with each model of mathematical machine, therefore we define *abstract equations over languages*. We introduce two composition operators for abstract languages: synchronous composition, $\bullet$, and parallel composition, $\diamond$, and we check conformity by language containment.

A key contribution is the *computation of the most general solutions of the language equations* $A \bullet X \subseteq C$ and $A \diamond X \subseteq C$, found respectively as $S = \overline{A \bullet \overline{C}}$, and $S = \overline{A \diamond \overline{C}}$. The derivation sheds lights on the properties required of a composition operator to yield such a closed formula as largest solution, and explains when different equations give rise to that same type of solution formula. These formulas turn out to subsume a panoply of specialized solutions derived in the past for specific composition operators and topologies.

Then we specialize such language equations to languages associated with chosen classes of automata used for modeling hardware and software systems, namely, regular languages as counterparts of finite automata, FSM languages as counterparts of FSMs. Thus we can operate algorithmically on those languages through their automata and study how to solve effectively their related language equations. It is important to find solutions within the same language class of the equation, e.g., when studying FSM language equations we look for solutions that are FSM languages. Moreover, we are interested in subsets of solutions characterized by further properties of practical interest, e.g., FSM languages that satisfy the Moore property; thus the valid solutions are restricted further.

Various contributions, investigating partial aspects of the topic of this research, have been published. A complete survey is provided in Sec. 6 (see also [15], Chap. 6, and [13]). A direct antecedent of this work is [29] on FSM equations under parallel composition (called "asynchronous equations"). A subset of the material on parallel language equations has been reported at a conference [39].

2 Equations over Languages

2.1 Languages and Operators

Definition 2.1 *An alphabet is a finite set of symbols. The set of all finite strings over a fixed alphabet X is denoted by X^*. X^* includes the empty string ϵ. A subset $L \subseteq X^*$ is called a* **language** *over alphabet X.*

Some standard operations on languages are:

1. Given languages L_1 and L_2, respectively over alphabets X_1 and X_2, the language $L_1 \cup L_2$ over alphabet $X_1 \cup X_2$ is the **union** of languages L_1 and L_2.

2. Given languages L_1 and L_2, respectively over alphabets X_1 and X_2, the language $L_1 L_2 = \{\alpha\beta \mid \alpha \in L_1, \beta \in L_2\}$ over alphabet $X_1 \cup X_2$ is the **concatenation** of languages L_1 and L_2. Define $L^0 = \{\epsilon\}$, $L^i = LL^{i-1}$. The **Kleene closure** of L is the set $L^* = \cup_{i=0}^{\infty} L^i$ and the **positive Kleene closure** of L is $L^+ = \cup_{i=1}^{\infty} L^i$. Finally, the **$l$-bounded Kleene closure** of L is set $L^{\leq l} = \cup_{i=0}^{l} L^i$.

3. Given languages L_1 and L_2, respectively over alphabets X_1 and X_2, the language $L_1 \cap L_2$ over alphabet $X_1 \cap X_2$ is the **intersection** of languages L_1 and L_2. If $X_1 \cap X_2 = \emptyset$ then $L_1 \cap L_2 = \emptyset$.

4. Given a language L over alphabet X, the language $\overline{L} = X^* \setminus L$ over alphabet X is the **complement** of language L. Similarly, given languages L_1 and L_2, respectively over alphabets X_1 and X_2, the language $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$ over alphabet X_1 is the **difference** of languages L_1 and L_2.

5. Given a language L over alphabet X, the language of all prefixes of words in L is $\mathit{Init}(L) = \{x \in X^* \mid \exists y \in X^*, xy \in L\}$.

It is useful to recall the notions of substitution and homomorphism of languages [14]. A **substitution** f is a mapping of an alphabet Σ onto subsets of Δ^* for some alphabet Δ. The substitution f is extended to strings by setting $f(\epsilon) = \epsilon$ and $f(xa) = f(x)f(a)$. An **homomorphism** h is a substitution such that $h(a)$ is a single string for each symbol a in the alphabet Σ. We introduce some often useful operations on languages.

1. Given a language L over alphabet $X \times V$, consider the homomorphism $p : X \times V \to V$ defined as

$$p((x, v)) = v,$$

then the language

$$L_{\downarrow V} = \{p(\alpha) \mid \alpha \in L\}$$

over alphabet V is the **projection** of language L to alphabet V, or V-projection of L. By definition of substitution $p(\epsilon) = \epsilon$.

2. Given a language L over alphabet X and an alphabet V, consider the substitution $l : X \to X \times V$ defined as

$$l(x) = \{(x, v) \mid v \in V\},$$

then the language

$$L_{\uparrow V} = \{l(\alpha) \mid \alpha \in L\}$$

over alphabet $X \times V$ is the **lifting** of language L to alphabet V, or V-lifting of L. By definition of substitution $l(\epsilon) = \epsilon$.

3. Given a language L over alphabet $X \cup V$, consider the homomorphism $r : X \cup V \to V$ defined as

$$r(y) = \begin{cases} y & \text{if } y \in V \\ \epsilon & \text{if } y \in X \setminus V \end{cases},$$

then the language

$$L_{\Downarrow V} = \{r(\alpha) \mid \alpha \in L\}$$

over alphabet V is the **restriction** of language L to alphabet V, or V-restriction of L, i.e., words in $L_{\Downarrow V}$ are obtained from those in L by deleting all the symbols in X that are not symbols in V. By definition of substitution $r(\epsilon) = \epsilon$.

4. Given a language L over alphabet X and an alphabet V disjoint from X, consider the mapping $e : X \to X \cup V$ defined as

$$e(x) = \{\alpha x \beta \mid \alpha, \beta \in V^*\},$$

then the language

$$L_{\Uparrow V} = \{e(\alpha) \mid \alpha \in L\}$$

over alphabet $X \cup V$ is the **expansion** of language L to alphabet V, or V-expansion of L, i.e., words in $L_{\Uparrow V}$ are obtained from those in L by inserting anywhere in them words from V^*. Notice that e is not a substitution and that $e(\epsilon) = \{\alpha \mid \alpha \in V^*\}$.

Given a language L over alphabet X, an alphabet V disjoint from X and a natural number l, consider the mapping $e_l : X \to X \cup V$ defined as

$$e_l(x) = \{\alpha x \beta \mid \alpha, \beta \in V^{\leq l}\},$$

then the language

$$L_{\Uparrow(V,l)} = \{e_l(\alpha) \mid \alpha \in L\}$$

over alphabet $X \cup V$ is the **l-bounded expansion** of language L over alphabet V, or (V,l)-expansion of L, i.e., words in $L_{\Uparrow V}$ are obtained from those in L by inserting anywhere in them words from $V^{\leq l}$. Notice that e_l is not a substitution and that $e_l(\epsilon) = \{\alpha \mid \alpha \in V^{\leq l}\}$.

By definition $\emptyset_{\downarrow V} = \emptyset, \emptyset_{\uparrow V} = \emptyset, \emptyset_{\Downarrow V} = \emptyset, \emptyset_{\Uparrow V} = \emptyset, \emptyset_{\Uparrow(V,l)} = \emptyset$.

The following straightforward facts hold between the projection and lifting operators, and between the restriction and expansion operators. In the following, unless otherwise stated, the union is taken over non-disjoint alphabets.

Proposition 2.1 *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 $(L_{\downarrow X})_{\uparrow Y} \supseteq L$.
(c) Given alphabets X and Y (X, Y disjoint), and a language L over alphabet X, then $(L_{\Uparrow Y})_{\Downarrow X} = L$.
(d) Given alphabets X and Y (X, Y disjoint), and a language L over alphabet $X \cup Y$, then $(L_{\Downarrow X})_{\Uparrow Y} \supseteq L$.

Proposition 2.2 *Given alphabets X and Y, a language L over alphabet X and a string $\alpha \in (X \times Y)^*$, then $\alpha_{\downarrow X} \in L$ iff $\alpha \in L_{\uparrow Y}$.*

Given alphabets X and Y, a language L over alphabet X and a string $\alpha \in (X \cup Y)^*$, then $\alpha_{\Downarrow X} \in L$ iff $\alpha \in L_{\Uparrow Y}$.

Proposition 2.3 *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 M_1, M_2 be languages over alphabet $I \times U$. Then $\downarrow$ commutes with $\cup$

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

(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 $\cap$

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

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

($\Rightarrow$) If the string $(i_1,u_1)\ldots(i_k,u_k) \in (L_1 \cap L_2)_{\uparrow I}$, then $u_1 \ldots u_k \in L_1 \cap L_2$; thus $u_1 \ldots u_k \in L_1$, $u_1 \ldots u_k \in L_2$, and so $(i_1,u_1)\ldots(i_k,u_k) \in L_{1\uparrow I}$, $(i_1,u_1)\ldots(i_k,u_k) \in L_{2\uparrow I}$, implying $(i_1,u_1)\ldots(i_k,u_k) \in L_{1\uparrow I} \cap L_{2\uparrow I}$.

($\Leftarrow$) If the string $(i_1,u_1)\ldots(i_k,u_k) \in L_{1\uparrow I} \cap L_{2\uparrow I}$, then $(i_1,u_1)\ldots(i_k,u_k) \in L_{1\uparrow I}$, $(i_1,u_1)\ldots(i_k,u_k) \in L_{2\uparrow I}$; thus $u_1 \ldots u_k \in L_1$, $u_1 \ldots u_k \in L_2$, implying $u_1 \ldots u_k \in L_1 \cap L_2$, and so $(i_1,u_1)\ldots(i_k,u_k) \in (L_1 \cap L_2)_{\uparrow I}$.

Similarly one proves the first and third identity involving $\cup$.

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

($\Rightarrow$) If the string $u_1 \ldots u_k \in (M_1 \cap M_2)_{\downarrow U}$ then there exists $i_1 \ldots i_k$ such that $(i_1,u_1)\ldots(i_k,u_k) \in M_1 \cap M_2$, i.e., $(i_1,u_1)\ldots(i_k,u_k) \in M_1$, $(i_1,u_1)\ldots(i_k,u_k) \in M_2$, and so $u_1 \ldots u_k \in M_{1\downarrow U}$ and $u_1 \ldots u_k \in M_{2\downarrow U}$.

($\Leftarrow$) If the string $u_1 \ldots u_k \in M_{1\downarrow U} \cap M_{2\downarrow U}$, i.e., $u_1 \ldots u_k \in M_{1\downarrow U}$ and $u_1 \ldots u_k \in M_{2\downarrow U}$, then there exists $i_1 \ldots i_k$ such that $(i_1,u_1)\ldots(i_k,u_k) \in M_1$. Moreover, since $M_2 = (M_{2\downarrow U})_{\uparrow I}$, from $u_1 \ldots u_k \in M_{2\downarrow U}$ it follows that $(i_1,u_1)\ldots(i_k,u_k) \in M_2$. In summary, $(i_1,u_1)\ldots(i_k,u_k) \in M_1$ and $(i_1,u_1)\ldots(i_k,u_k) \in M_2$, implying $(i_1,u_1)\ldots(i_k,u_k) \in M_1 \cap M_2$, from which follows $u_1 \ldots u_k \in (M_1 \cap M_2)_{\downarrow U}$. □

Corollary 2.1 *Let $L_i, i = 1, \ldots, n$ be languages over alphabet U. Then $\uparrow$ commutes with both $\cup$ and $\cap$*

$$(\bigcup L_i)_{\uparrow I} = \bigcup (L_{i\uparrow I}),$$
$$(\bigcap L_i)_{\uparrow I} = \bigcap (L_{i\uparrow I}).$$

Let $M_i, i = 1, \ldots, n$ be languages over alphabet $I \times U$., Then $\downarrow$ commutes with $\cup$

$$(\bigcup M_i)_{\downarrow U} = \bigcup (M_{i\downarrow U}).$$

Let $M_i, i = 1, \ldots, n$ be languages over alphabet $I \times U$. If $M_2 = (M_{2\downarrow U})_{\uparrow I}, \ldots, M_n = (M_{n\downarrow U})_{\uparrow I}$ (or any collection of $n-1$ languages M_i satisfies this property), then $\downarrow$ commutes with $\cap$

$$(\bigcap M_i)_{\downarrow U} = \bigcap (M_{i\downarrow U}).$$

The proof is by induction based on Prop. 2.3.

Proposition 2.4 *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 M_1, M_2 be languages over alphabet $I \cup U$. Then $\Downarrow$ commutes with $\cup$

$$(M_1 \cup M_2)_{\Downarrow U} = M_{1\Downarrow U} \cup M_{2\Downarrow U}.$$

(d) Let M_1, M_2 be languages over alphabet $I \cup U$. If $M_2 = (M_{2\Downarrow U})_{\Uparrow I}$ (or $M_1 = (M_{1\Downarrow U})_{\Uparrow I}$) then $\Downarrow$ commutes with $\cap$

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

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

($\Rightarrow$) If the string $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in (L_1 \cap L_2)_{\Uparrow I}$ and $\alpha_1, \ldots, \alpha_k, \alpha_{k+1} \in I^*$, then $u_1 \ldots u_k \in L_1 \cap L_2$; thus $u_1 \ldots u_k \in L_1$, $u_1 \ldots u_k \in L_2$, and so $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_{1\Uparrow I}$, $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_{2\Uparrow I}$, implying $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_{1\Uparrow I} \cap L_{2\Uparrow I}$.

($\Leftarrow$) If the string $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_{1\Uparrow I} \cap L_{2\Uparrow I}$, then it holds $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_{1\Uparrow I}$ and $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_{2\Uparrow I}$; thus $u_1 \ldots u_k \in L_1$, $u_1 \ldots u_k \in L_2$, implying $u_1 \ldots u_k \in L_1 \cap L_2$, and so $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in (L_1 \cap L_2)_{\Uparrow I}$.

Similarly one proves the first and third identity involving $\cup$.

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

($\Rightarrow$) If the string $u_1 \ldots u_k \in (M_1 \cap M_2)_{\Downarrow U}$ then there exists $\alpha_1, \ldots \alpha_k, \alpha_{k+1} \in I^*$ such that the string $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_1 \cap M_2$, i.e., $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_1$, $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_2$, and so $u_1 \ldots u_k \in M_{1\Downarrow U}$ and $u_1 \ldots u_k \in M_{2\Downarrow U}$.

($\Leftarrow$) If the string $u_1 \ldots u_k \in M_{1\Downarrow U} \cap M_{2\Downarrow U}$, i.e., $u_1 \ldots u_k \in M_{1\Downarrow U}$ and $u_1 \ldots u_k \in M_{2\Downarrow U}$, then there exists $\alpha_1 \ldots \alpha_k \alpha_{k+1} \in I^*$ such that $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_1$. Moreover, since $M_2 = (M_{2\Downarrow U})_{\Uparrow I}$, from $u_1 \ldots u_k \in M_{2\Downarrow U}$ it follows that $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_2$. In summary, $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_1$ and $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_2$, implying $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_1 \cap M_2$, from which follows $u_1 \ldots u_k \in (M_1 \cap M_2)_{\Downarrow U}$. □

2.2 Finite Automata and Regular Expressions

Definition 2.2 *A* **finite automaton** *(FA) is defined as a 5-tuple $F = \langle S, \Sigma, \Delta, r, Q \rangle$. S represents the finite state space, Σ 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 $i \in \Sigma$ iff $(i, p, n) \in \Delta$. 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 ϵ-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 Σ^* (i.e., $\Delta \subseteq \Sigma^* \times S \times S$) as follows: $(\rho i, s, s'') \in \Delta$ iff 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 Δ can be replaced by the next state function δ, defined as $\delta : \Sigma \times S \rightarrow S$, where $n \in S$ is the next state of present state $p \in S$ on symbol $i \in \Sigma$ iff $n = \delta(i, p)$. 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 $\{x \mid \delta(x, r) = s\}$.*

The languages associated with finite automata are the regular languages, defined by means of regular expressions.

Definition 2.3 *The* **regular expressions** *over an alphabet Σ are defined recursively as follows:*

1. *$\emptyset$ is a regular expression and denotes the empty set.*
2. *ϵ is a regular expression and denotes the set $\{\epsilon\}$.*
3. *For each $a \in \Sigma$, a is a regular expression and denotes the set $\{a\}$.*
4. *If r and s are regular expressions denoting the languages R and S, respectively, then $(r + s)$, (rs) and (r^*) are regular expressions that denote the sets $R \cup S$, RS and R^*, respectively.*

The sets denoted by regular expressions are the **regular languages.**

Regular languages are closed under union, concatenation, complementation and intersection. Also regular languages are closed under projection, lifting and restriction, because they are closed under substitution [14]. Regular languages are closed under expansion, because Sec. 3.2 describes an algorithm that, given the finite automaton of a language, returns the finite automaton of the expanded language.

2.3 Classes of Languages

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

Definition 2.4 *A language L over alphabet X is* **prefix-closed** *if $\forall \alpha \in X^*\ \forall x \in X\ [\alpha x \in L \Rightarrow \alpha \in L]$. Equivalently, L is prefix-closed iff $L = Init(L)$.*

Definition 2.5 *A language L over alphabet $X = I \times O$ is* **I-progressive** *if*

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

Definition 2.6 *A language L over alphabet $I \times O$ is* $I_\downarrow$**-defined** *if* $L_{\downarrow I} = I^*$.

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

Example 2.1 *The language* $L = \{\epsilon + i_1o_1 + i_1o_2(i_1o_1)^*\}$ *is* $I_\downarrow$*-defined, but not* I*-progressive, as witnessed by* $\alpha = i_1o_1 \in L$ *and* $i = i_1$ *for which there is no* o *such that* $\alpha\, io \in L$.

Definition 2.7 *A language L over alphabet $X = I \times O$ is* **Moore** *with respect to alphabet I, if*

$$\forall \alpha \in L\, \forall (i,o) \in X\, \forall (i',o') \in X\, [\alpha\,(i,o) \in L \Rightarrow [\alpha\,(i',o') \in L \Rightarrow \alpha\,(i',o) \in L]].$$

Definition 2.8 *A language $L \subseteq (IO)^*$ over alphabet $I \cup O$ (I and O disjoint) is* IO**-prefix-closed** *if*

$$\forall \alpha \in (IO)^*\, \forall io \in IO\, [\alpha\, io \in L \Rightarrow \alpha \in L].$$

Definition 2.9 *A language $L \subseteq (IO)^*$ over alphabet $I \cup O$ (I and O disjoint) is* IO**-progressive** *if*

$$\forall \alpha \in (IO)^*\, \forall i \in I\, \exists o \in O\, [\alpha \in L \Rightarrow \alpha\, io \in L].$$

Definition 2.10 *A language $L \subseteq (IU^*O)^*$ over alphabet $I \cup U \cup O$ (I, U and O disjoint) is* I^*O**-progressive** *if*

$$\forall \alpha \in (IU^*O)^*\, \forall i \in I\, \exists \beta \in U^*\, \exists o \in O\, [\alpha \in L \Rightarrow \alpha\, i\beta o \in L].$$

Example 2.2 *a) Let $I = \{i_1, i_2\}$, $O = \{o_1, o_2\}$ and $U = \{u_1, u_2\}$. The language $L = \{(i_1u_1u_2^*u_1o_1 + i_2u_1^*o_2)^*\}$ is I^*O-progressive, since any word in L can be extended to a word in L by suffixes starting with either i_1 or i_2. The corresponding automaton is shown in Fig. 2(a).*

b) Let $I = \{i_1, i_2\}$, $O = \{o_1, o_2\}$ and $U = \{u_1\}$. The language $L = \{(i_1o_1)^* + (i_1o_1)^*i_2u_1^*o_2(i_1u_1^*o_2)^*\}$ is not I^*O-progressive, since the words in the set $\{i_2u_1^*o_2(i_1u_1^*o_2)^*\}$ are in L, but when $i = i_2$ there is no $\beta \in U^*$ and no $o \in O$ such that $\alpha\, i\beta o \in L$ (e.g., $\alpha = i_2u_1o_2$ cannot be extended by any suffix starting with i_2). The corresponding automaton is shown in Fig. 2(b).

Definition 2.11 *A language L over alphabet $I \cup O$ (I and O disjoint) is* $I_\Downarrow$**-defined** *if* $L_{\Downarrow I} = I^*$.

An IO-progressive language is $I_\Downarrow$-defined, so is an I^*O-progressive language, but the converse does not hold.

Definition 2.12 *A language L over alphabet $X \cup U$ (X and U disjoint) is* U**-deadlock-free** *if*

$$\forall \alpha \in (X \cup U)^*\, \forall u \in U\, \exists \beta \in U^*\, \exists x \in X\, [\alpha u \in L \Rightarrow \alpha u\, \beta x \in L].$$

Any language $L \subseteq (IU^*O)^*$ is U-deadlock-free (because no word ending by a symbol $u \in U$ belongs to the language).

Example 2.3 *a) Let $X = I \cup O$, $I = \{i_1, i_2\}$, $O = \{o_1, o_2\}$ and $U = \{u_1, u_2\}$. The language $L = \{(i_1(u_1u_2^*u_1)^*o_1)^* + (i_1(u_1u_2^*u_1)^*o_1)^*i_1u_1u_2^*\}$ is U-deadlock-free, because any word in the language terminating by u_1 or u_2 can be extended by suffix u_1 to a word in the language terminating by o_1. The corresponding automaton is shown in Fig. 2(c).*

b) Let $X = I \cup O$, $I = \{i_1, i_2\}$, $O = \{o_1, o_2\}$ and $U = \{u_1, u_2, u_3\}$. The language $L = \{i_1(u_1u_2^*u_3)^*o_1)^* + (i_1(u_1u_2^*u_3)^*o_1)^*i_1u_1u_2^* + (i_1(u_1u_2^*u_3)^*o_1)^*i_1u_1u_2^*u_1u_2^*\}$ is not U-deadlock-free, since the words in the set $\{(i_1(u_1u_2^*u_3)^*o_1)^*i_1u_1u_2^*u_1u_2^*\}$ cannot be extended to words in L (e.g., $\alpha = i_1u_1u_2u_1$). The corresponding automaton is shown in Fig. 2(d).

Definition 2.13 *A language L over alphabet $X \cup U$ (X and U disjoint) is* U**-convergent** *if* $\forall \alpha \in X^*$ *the language* $\alpha_{\Uparrow U} \cap L$ *is finite, otherwise it is* U**-divergent.**

Example 2.4 *The language $L = \{iu^*o\}$ where $X = \{i, o\}$ and $U = \{u\}$ is U-divergent, as witnessed by the string $\alpha = io \in X$ whose expansion includes the infinite set $\{iu^*o\}$ coinciding with L: $\{\alpha_{\Uparrow U}\} = \{(io)_{\Uparrow\{u\}}\} = \{u^*iu^*vu^*\} \supset \{iu^*v\} = L$.*

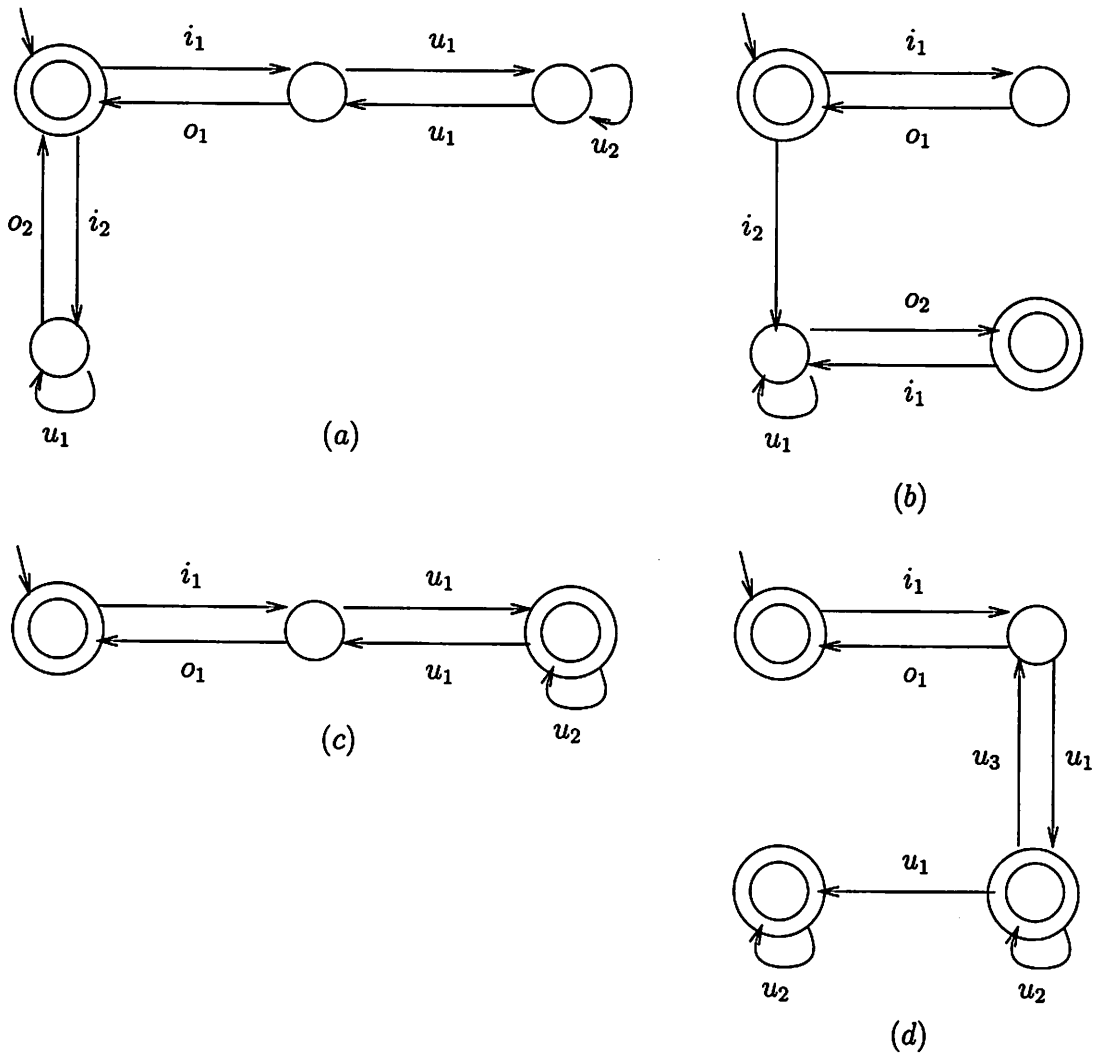

Figure 2: (a) Finite automaton of the language described in Example 2.2-a); (b) Finite automaton of the language described in Example 2.2-b); (c) Finite automaton of the language described in Example 2.3-a); (d) Finite automaton of the language described in Example 2.3-b).

2.4 Composition of Languages

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

Definition 2.14 *Given the disjoint alphabets I, U, O, language L_1 over $I \times U$ and language L_2 over $U \times O$, the* **synchronous composition** *of languages L_1 and L_2 is the language* [1] $[(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}]_{\downarrow I \times O}$, *denoted by $L_1 \bullet L_2$, defined over $I \times O$.*

Definition 2.15 *Given the disjoint alphabets I, U, O, language L_1 over $I \cup U$ and language L_2 over $U \cup O$, the* **parallel composition** *of languages L_1 and L_2 is the language $[(L_1)_{\Uparrow O} \cap (L_2)_{\Uparrow I}]_{\Downarrow I \cup O}$, denoted by $L_1 \diamond L_2$, defined over $I \cup O$.*

Given alphabets I, U, O, language L_1 over $I \cup U$ and language L_2 over $U \cup O$, the **l-bounded parallel composition** *of languages L_1 and L_2 is the language $[(L_1)_{\Uparrow O} \cap (L_2)_{\Uparrow I} \cap (I \cup O)^*_{\Uparrow (U,l)}]_{\Downarrow I \cup O}$, denoted by $L_1 \diamond_l L_2$, defined over $I \cup O$.*

By definition of the operations $\downarrow V$, $\uparrow V$, $\Downarrow V$, $\Uparrow V$, $\Uparrow (V,l)$ it follows that $\emptyset \bullet L = L \bullet \emptyset = \emptyset$, $\emptyset \diamond L = L \diamond \emptyset = \emptyset$, $\emptyset \diamond_l L = L \diamond_l \emptyset = \emptyset$.

When $l = \infty$ the definition of l-bounded parallel composition reduces to the definition of parallel composition of languages, because then $(I \cup O)^*_{\Uparrow (U,l)}$ becomes $(I \cup O \cup U)^*$, that is the universe over $I \cup O \cup U$, and so it can be dropped from the conjunction.

Variants of *synchronous composition* are introduced in [6] as *product*, $\times$ (with the comment *sometimes called completely synchronous composition*), and in [21] as *synchronous parallel composition*, $\otimes$. Variants of *parallel composition* are introduced in [6] as *parallel composition*, $\|$ (with the comment *often called synchronous composition*), and in [21] as *interleaving parallel composition*, $\|$; the same operator was called *asynchronous composition* in [29]. These definitions were usually introduced for regular languages; actually they were more commonly given for finite automata.

It has also been noticed by Kurshan [21] and Arnold [1] that asynchronous systems can also be modeled with the synchronous interpretation, using null transitions to keep a transition system in the same state for an arbitrary period of time. Kurshan [21] observes that: "While synchronous product often is thought to be a simple -even uninteresting!- type of coordination, it can be shown that, through use of nondeterminism, this conceptually simple coordination serves to model the most general 'asynchronous' coordination, i.e., where processes progress at arbitrary rates relative to one another. In fact the 'interleaving' model, the most common model for asynchrony in the software community, can be viewed as a special case of this synchronous product." A technical discussion can be found in [22].

In the sequel it will be useful to extend some properties of languages to the composition of two languages. As examples, we illustrate the extension for I-progressive and I^*O-progressive languages.

Definition 2.16 *Given a language A over alphabet $I \times U$, a language B over alphabet $U \times O$ is A-***compositionally I-progressive** *if the language $L = A_{\uparrow O} \cap B_{\uparrow I}$ over alphabet $X = I \times U \times O$ is I-progressive, i.e., $\forall \alpha \in X^* \; \forall i \in I \; \exists (u,o) \in U \times O \; [\alpha \in L \Rightarrow \alpha\,(i,u,o) \in L]$.*

Definition 2.17 *Given a language A over alphabet $I \cup U$, a language B over alphabet $U \cup O$ is A-***compositionally I^*O-progressive** *if the language $L = A_{\uparrow O} \cap B_{\uparrow I} \subseteq (IU^*O)^*$ over alphabet $X = I \cup U \cup O$ (I, U and O disjoint) is I^*O-progressive, i.e., $\forall \alpha \in (IU^*O)^* \; \forall i \in I \; \exists \beta \in U^* \; \exists o \in O \; [\alpha \in L \Rightarrow \alpha\, i \beta o \in L]$.*

Defn. 2.17 ensures that the composition does not fall into a deadlock.

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

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

2.5 Solution of Equations over Languages

2.5.1 Language Equations under Synchronous Composition

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

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

Definition 2.18 *Given alphabets I, U, O, a language A over alphabet $I \times U$ and a language C over alphabet $I \times O$, language B over alphabet $U \times O$ is called a* **solution** *of the equation $A \bullet X \subseteq C$ iff $A \bullet B \subseteq C$. The* **largest solution** *is the solution that contains any other solution. $B = \emptyset$ is the trivial solution.*

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

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

$$\begin{aligned}
A \bullet \{\alpha\} \subseteq C &\Leftrightarrow \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I})_{\downarrow I \times O} \cap \overline{C} = \emptyset &\Leftrightarrow \quad \textit{by Prop. 2.1(a)}\ \overline{C} = (\overline{C}_{\uparrow U})_{\downarrow I \times O} \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I})_{\downarrow I \times O} \cap (\overline{C}_{\uparrow U})_{\downarrow I \times O} = \emptyset &\Leftrightarrow \quad \textit{by Prop. 2.3(d) since}\ ((\overline{C}_{\uparrow U})_{\downarrow I \times O})_{\uparrow U} = \overline{C}_{\uparrow U} \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U})_{\downarrow I \times O} = \emptyset &\Leftrightarrow \\
A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U} = \emptyset &\Leftrightarrow \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O} = \emptyset &\Leftrightarrow \quad \textit{by Prop. 2.3(d) since}\ \{\alpha\}_{\uparrow I} = (\{\alpha\}_{\uparrow I})_{\downarrow U \times O})_{\uparrow I} \\
(\{\alpha\}_{\uparrow I})_{\downarrow U \times O} \cap (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O} = \emptyset &\Leftrightarrow \quad \textit{by Prop. 2.1(a)}\ (\{\alpha\}_{\uparrow I})_{\downarrow U \times O} = \{\alpha\} \\
\{\alpha\} \cap (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O} = \emptyset &\Leftrightarrow \\
\alpha \notin (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O} &\Leftrightarrow \\
\alpha \in \overline{(A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O}} &\Leftrightarrow \\
\alpha \in \overline{A \bullet \overline{C}}
\end{aligned}$$

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

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

□

Corollary 2.2 *A language B over alphabet $U \times O$ is a solution of $A \bullet X \subseteq C$ iff $B \subseteq \overline{A \bullet \overline{C}}$.*

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

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

Proposition 2.5 *If Pref$(S) \neq \emptyset$, then Pref(S) is the largest prefix-closed solution of the equation $A \bullet X \subseteq C$.*
If Pref$(S) = \emptyset$, then the equation $A \bullet X \subseteq C$ has no prefix-closed solution.
If the language S does not include the empty string, then $A \bullet X \subseteq C$ has no prefix-closed solution.

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

Proposition 2.6 *If $Prog(S) \neq \emptyset$, then the language $Prog(S)$ is the largest U-progressive solution of the equation $A \bullet X \subseteq C$.*
If $Prog(S) = \emptyset$, then the equation $A \bullet X \subseteq C$ has no U-progressive solution.

2.5.2 Language Equations under Parallel Composition

Given pairwise disjoint alphabets I, U, O, a language A over alphabet $I \cup U$ and a language C over alphabet $I \cup O$, consider the language equation

$$A \diamond X \subseteq C. \tag{3}$$

Definition 2.19 *Given pairwise disjoint alphabets I, U, O, a language A over alphabet $I \cup U$ and a language C over alphabet $I \cup O$, language B over alphabet $U \cup O$ is called a* **solution** *of the equation $A \diamond X \subseteq C$ iff $A \diamond B \subseteq C$. The* **largest solution** *is a solution that contains any other solution. $B = \emptyset$ is the trivial solution.*

Theorem 2.2 *The largest solution of the equation $A \diamond X \subseteq C$ is the language $S = \overline{A \diamond \overline{C}}$, if $S \neq \emptyset$.*

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

$$\begin{aligned}
A \diamond \{\alpha\} \subseteq C &\Leftrightarrow \\
(A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I})_{\Downarrow IUO} \cap \overline{C} = \emptyset &\Leftrightarrow \text{ by Prop. 2.1(c) } \overline{C} = (\overline{C}_{\Uparrow U})_{\Downarrow IUO} \\
(A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I})_{\Downarrow IUO} \cap (\overline{C}_{\Uparrow U})_{\Downarrow IUO} = \emptyset &\Leftrightarrow \text{ by Prop. 2.4(d) since } ((\overline{C}_{\Uparrow U})_{\Downarrow IUO})_{\Uparrow U} = \overline{C}_{\Uparrow U} \\
(A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap \overline{C}_{\Uparrow U})_{\Downarrow IUO} = \emptyset &\Leftrightarrow \\
A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap \overline{C}_{\Uparrow U} = \emptyset &\Leftrightarrow \\
(A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap \overline{C}_{\Uparrow U})_{\Downarrow UUO} = \emptyset &\Leftrightarrow \text{ by Prop. 2.4(d) since } \{\alpha\}_{\Uparrow I} = ((\{\alpha\}_{\Uparrow I})_{\Downarrow UUO})_{\Uparrow I} \\
(\{\alpha\}_{\Uparrow I})_{\Downarrow UUO} \cap (A_{\Uparrow O} \cap \overline{C}_{\Uparrow U})_{\Downarrow UUO} = \emptyset &\Leftrightarrow \text{ by Prop. 2.1(c) } (\{\alpha\}_{\Uparrow I})_{\Downarrow UUO} = \{\alpha\} \\
\{\alpha\} \cap (A_{\Uparrow O} \cap \overline{C}_{\Uparrow U})_{\Downarrow UUO} = \emptyset &\Leftrightarrow \\
\alpha \notin (A_{\Uparrow O} \cap \overline{C}_{\Uparrow U})_{\Downarrow UUO} &\Leftrightarrow \\
\alpha \in \overline{(A_{\Uparrow O} \cap \overline{C}_{\Uparrow U})_{\Downarrow UUO}} &\Leftrightarrow \\
\alpha \in \overline{A \diamond \overline{C}}
\end{aligned}$$

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

$$S = \overline{A \diamond \overline{C}}. \tag{4}$$

□

Corollary 2.3 *A language B over alphabet $U \cup O$ is a solution of $A \diamond X \subseteq C$ iff $B \subseteq \overline{A \diamond \overline{C}}$.*

Proposition 2.7 *If S is U-convergent, then S is the largest U-convergent solution of the equation, and a language $B \neq \emptyset$ is a U-convergent solution iff $B \subseteq S$.*

When S is not U-convergent the largest U-convergent solution does not exist, and any finite subset of S is a U-convergent solution. An analogous proposition and remark hold for S-compositionally U-convergent solutions.

2.5.3 Language Equations under Bounded Parallel Composition

Theorem 2.3 *The largest solution of the equation* $A \diamond_l X \subseteq C$ *is the language*

$$S = \overline{(A_{\Uparrow O} \cap \overline{C}_{\Uparrow(U,I)})_{\Downarrow U \cup O}}.$$

Proof.

$$\begin{aligned} A \diamond_l \{\alpha\} \subseteq C &\Leftrightarrow \\ (A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap (I \cup O)^*_{\Uparrow(U,I)})_{\Downarrow I \cup O} \cap \overline{C} = \emptyset &\Leftrightarrow \\ A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap \overline{C}_{\Uparrow(U,I)} = \emptyset &\Leftrightarrow \\ \alpha \notin (A_{\Uparrow O} \cap \overline{C}_{\Uparrow(U,I)})_{\Downarrow U \cup O} &\Leftrightarrow \\ \alpha \in \overline{(A_{\Uparrow O} \cap \overline{C}_{\Uparrow(U,I)})_{\Downarrow U \cup O}} & \end{aligned}$$

□

3 Equations over Finite Automata

3.1 Equations over Mathematical Machines

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

3.2 Solution of Equations over Regular Languages

Two well-known results [14] are that non-deterministic finite automata are equivalent (w.r. to language equality) to deterministic ones and that regular expressions are equivalent to finite automata. By applying the algorithm of subset construction one converts a NDFA into an equivalent DFA (complete by construction). Given an NDFA $F = \langle S, \Sigma, \Delta, r, Q \rangle$, the process of subset construction builds the DFA $F' = \langle 2^S, \Sigma, \delta, r, Q' \rangle$, where 1) the states $\tilde{s} \in 2^S$ are the subsets of S, 2) the transition relation is $\delta(i, \tilde{s}) = \cup_{s \in \tilde{s}} \{s' \mid (i, s, s') \in \Delta\}$ and 3) a state is final, i.e., $\tilde{s} \in Q' \subseteq 2^S$, iff $\tilde{s} \cap Q \neq \emptyset$. Since many of the states in 2^S are unreachable from the initial state, they can be deleted and so the determinized automaton usually has fewer states than the power set. To make a NDFA complete it is not necessary to apply the full-blown subset construction, but it suffices to add a new non-accepting state s_d whose incoming transitions are (i, s, s_d) for all i, s for which there was no transition in the original automaton. By a closure construction [14], an NDFA with ϵ-moves can be converted to an NDFA without ϵ-moves; subset construction must be applied at the end to determinize it.

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 [14]. Here we sketch the procedures for projection, lifting, restriction and expansion:

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 replacing each edge $((x,v),s,s')$ by the edge (x,s,s') and then applying subset construction to determinize it.

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

restriction ($\Downarrow$) Given FA F that accepts language L over $X \cup V$, FA F' that accepts language $L_{\Downarrow V}$ over V is obtained from F by the following procedure:

1. $\forall x \in X \setminus V$, change every edge (x,s,s') into the edge (ϵ,s,s'), i.e., replace the symbols $x \in X \setminus V$ by ϵ.
2. Apply the closure construction to obtain an equivalent deterministic finite automaton without ϵ-moves (by a procedure similar to the subset construction, where the states of the final automaton are subsets of the states of the original automaton, and ϵ-moves are handled by the mechanism of ϵ-closure [14]).

expansion ($\Uparrow$) Given FA F that accepts language L over X, FA F' that accepts language $L_{\Uparrow V}$ over $X \cup V$ ($X \cap V = \emptyset$) is obtained from F by adding for each state s, $\forall v \in V$, the edge (self-loop) (v,s,s).

l-expansion ($\Uparrow_l$) Given FA F that accepts language L over X, FA F' that accepts language $L_{\Uparrow(V,l)}$, l integer, over $X \cup V$ ($X \cap V = \emptyset$) is obtained from F by the following procedure:

1. The set of states S' of F' is given by
$$S' = S \cup \{(s,j) \mid s \in S, 1 \leq j \leq l\}.$$
2. The next state relation Δ' of F' is given by
$$\begin{aligned} \Delta' \;=\; & \Delta \cup \{(v,s,(s,1)) \mid v \in V, s \in S\} \\ & \cup \{(v,(s,j),(s,j+1)) \mid v \in V, s \in S, 1 \leq j < l\} \\ & \cup \{(x,(s,j),s') \mid (x,s,s') \in \Delta, 1 \leq j \leq l\}. \end{aligned}$$
3. $r' = r$ and $Q' = Q$.

The procedures for projection, lifting and restriction guarantee the substitution property $f(\epsilon) = \epsilon$.

Given that all the operators used to express the solution of regular language equations have constructive counterparts on automata, we conclude that there is an effective way to solve equations over regular languages.

As an example, given a regular language equation $A \bullet X \subseteq C$, where A is a regular language over alphabet $I \times U$, C is over $I \times O$, and the unknown regular language X is over $U \times O$, an algorithm to build X follows.

Procedure 3.1 *Input: Regular language equation $A \bullet X \subseteq C$; Output: Largest regular language solution X*

1. Consider the finite automata $F(A)$ and $F(C)$ corresponding, respectively, to regular languages A and C.
2. Determinize $F(C)$ by subset construction, if it is a NDFA. The automaton $F(\overline{C})$ of $\overline{C}$ is obtained by interchanging the sets of accepting and non-accepting states of the determinization of $F(C)$.
3. Lift the language A to O by replacing each label (i,u) of a transition of $F(A)$ with all triples (i,u,o), $o \in O$. Lift the language $\overline{C}$ to U by replacing each label (i,o) of a transition of $F(\overline{C})$ with all triples (i,u,o), $u \in U$.

4. Build the automaton $F(A \cap \overline{C})$ of the intersection $A \cap \overline{C}$. The states are pairs of states of the lifted automata $F(A)$ and $F(\overline{C})$, the initial state is the pair of initial states, and a state of the intersection is accepting if both states of the pair are accepting. There is a transition from the state (s_1, s_2) to the state (s_1', s_2') labelled with action (i, u, o) in $F(A \cap \overline{C})$, if there are corresponding transitions labelled with (i, u, o) from state s_1 to state s_1' in $F(A)$ and from s_2 to s_2' in $F(\overline{C})$.

5. Project $F(A \cap \overline{C})$ to $U \times O$ to obtain $F(A \bullet \overline{C})$ by deleting i from the labels (i, u, o). Projection in general makes the finite automaton non-deterministic.

6. Determinize $F(A \bullet \overline{C})$ by subset construction, if it is a NDFA. The automaton $F(\overline{A \bullet \overline{C}})$ corresponding to the regular language solution $X = \overline{A \bullet \overline{C}}$ is obtained by interchanging the sets of accepting and non-accepting states of the determinization of $F(A \bullet \overline{C})$.

Notice that Proc. 3.1 holds for any regular language, not only for prefix-closed languages as in restricted versions reported in the literature.

A companion procedure to solve the regular language equation under parallel composition $A \diamond X \subseteq C$ is obtained from Proc. 3.1, after replacing the cartesian product with union, projection with restriction and lifting with expansion. The largest solution of parallel equations for prefix-closed regular languages had been known already in the process algebra literature [25, 32, 24],

4 Equations over Finite State Machines

4.1 Finite State Machines

Definition 4.1 *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 iff $(i, p, n, o) \in T$. State $r \in S$ represents the initial or reset state. We denote the restriction of relation T to $I \times S \times S$ (next state relation) by $T_n \subseteq I \times S \times S$, i.e., $(i, s, s') \in T_n \Leftrightarrow \exists o\, (i, s, s', o) \in T$; similarly, we denote the restriction of relation T to $I \times S \times O$ (output relation) by $T_o \subseteq I \times S \times O$, i.e., $(i, s, o) \in T_o \Leftrightarrow \exists s'\, (i, s, s', o) \in T$. We may use δ instead of T_n and λ instead of T_o. If at least one transition is specified for each present state and input pair, the FSM is said to be* **complete**. *If no transition is specified for at least one present state and input pair, the FSM is said to be* **partial**. *An FSM is said to be* **trivial** *when $T = \emptyset$, denoted by M_ϵ.*

It is convenient to think of the relations T_n and T_o as functions $T_n : I \times S \rightarrow 2^S$ and $T_o : I \times S \rightarrow 2^O$.

Definition 4.2 *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 4.3 *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).*

In a DFSM the next state n and the output o can be given, respectively, by a next state function $n = T_n(i, p)$ and an output function $o = T_o(i, p)$.

Definition 4.4 *An NDFSM is a* **pseudo non-deterministic FSM** *(PNDFSM)* [38], *or* **observably non-deterministic FSM** [8], *or* **observable FSM** [34], *if for each triple $(i, p, o) \in I \times S \times O$, there is at most one state n such that $(i, p, n, o) \in T$.*

The qualification "non-deterministic" is because for a given input and present state, there may be more than one possible output; however, edges (i.e., transitions) carrying different outputs must go to different next states. The further qualification "pseudo" non-deterministic is because its underlying finite automaton is deterministic. In a PNDFSM the next state n, if it exists, is unique for a given combination of input, present state and output, so it can be given by a partial next state function $n = T_n(i, p, o)$. Since the output is non-deterministic in general, it is represented by a relation $T_o \subseteq I \times S \times O$.

Definition 4.5 *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* $(i', p, n', o) \in T$ [2].

The transition relation T of an FSM can be extended in the usual way to a relation on $I^* \times S \times S \times O^*$: given a present state p and an input sequence $i_1 \dots i_k \in I^*$, $(i_1 \dots i_k, p, n, o_1 \dots o_k) \in T$ iff there is a sequence $s_1 \dots s_{k+1}$ such that $s_1 = p, \dots, s_{k+1} = n$ and for each $j = 1, \dots, k$ it holds that $(i_j, s_j, s_{j+1}, o_j) \in T$. A similar extension can be defined for T_p and T_n.

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

4.2 Languages of FSMs

We now introduce the notion of a language associated to an FSM. This is achieved by looking to the automaton underlying a given FSM. For our purposes, we define two related languages: one over the alphabet $I \times O$ and the other over the alphabet $I \cup O$, naturally associated, respectively, with synchronous and parallel composition, as it will be seen later.

For a language over $I \times O$, the automaton coincides with the original FSM where all states are made accepting and the edges carry a label of the type (i, o).

For a language over $I \cup O$, the automaton is obtained from the original FSM, by replacing each edge (i, s, s', o) by the pair of edges $(i, s, (s, i))$ and $(o, (s, i), s')$ where (s, i) is a new node (non-accepting state). All original states are made accepting. The automaton is deterministic because from (i, s, s'_1, o_1) and (i, s, s'_2, o_2) the edges $(i, s, (s, i))$, $(o_1, (s, i), s'_1)$ and $(o_2, (s, i), s'_2)$ are obtained (the same edge $(i, s, (s, i))$ works in both cases).

Definition 4.6 *Given an FSM* $M = \langle S, I, O, T, r \rangle$, *consider the finite automaton* $F(M) = \langle S, I \times O, \Delta, r, S \rangle$, *where* $((i, o), s, s') \in \Delta$ *iff* $(i, s, s', o) \in T$. *The language accepted by* $F(M)$ *is denoted* $L^{\times}_r(M)$, *and by definition is the* $\times$**-language** *of* M *at state* r. *Similarly* $L^{\times}_s(M)$ *denotes the language accepted by* $F(M)$ *when started at state* s, *and by definition is the* $\times$*-language of* M *at state* s.

Definition 4.7 *Given an FSM* $M = \langle S, I, O, T, r \rangle$, *consider the finite automaton* $F(M) = \langle S \cup (S \times I), I \cup O, \Delta, r, S \rangle$, *where* $(i, s, (s, i)) \in \Delta \wedge (o, (s, i), s') \in \Delta$ *iff* $(i, s, s', o) \in T$. *The language accepted by* $F(M)$ *is denoted* $L^{\cup}_r(M)$, *and by definition is the* $\cup$**-language** *of* M *at state* r. *Similarly* $L^{\cup}_s(M)$ *denotes the language accepted by* $F(M)$ *when started at state* s, *and by definition is the* $\cup$*-language of* M *at state* s. *By construction,* $L^{\cup}_s(M) \subseteq (IO)^*$, *where* IO *denotes the set* $\{io \mid i \in I, o \in O\}$.

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

Definition 4.8 *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** [3].

Remark When convenient, we will say that FSM M has property X if its associated FSM language has property X.

[2] Notice that this definition allows for NDFSMs of Moore type, contrary to the more common definition of Moore type: for each present state p there is an output o such that all transitions whose present state is p carry the same output o.

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

Definition 4.9 *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 $t \preceq s$, iff $L_t(M_B) \subseteq L_s(M_A)$. States t and s are* **equivalent states**, *written $t \cong s$, iff $t \preceq s$ and $s \preceq t$, i.e., when $L_t(M_B) = L_s(M_A)$. An FSM with no two equivalent states is a* **reduced** *FSM.*

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

For complete DFSMs reduction and equivalence of states coincide. Given an FSM language, there is a family of equivalent FSMs associated with it; for simplicity we will usually speak of the FSM associated with a given FSM language. In this paper, FSMs are assumed to be reduced, unless stated otherwise.

An FSM language is regular, whereas the converse is not true.

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

Notice that the coincidence of the notions of complete FSM and I-progressive associated language is due to the fact that FSMs are assumed to be PNDFSMs, i.e., their underlying automaton is deterministic, therefore a word has a unique run (sequence of transitions), from which an extension is possible under any input.

Theorem 4.2 *A regular language over alphabet $I \cup O$ is the language of a complete FSM over input alphabet I and output alphabet O iff $L \subseteq (IO)^*$, L is IO-prefix-closed and IO-progressive. A regular language $L \subseteq (IO)^*$ that is IO-prefix-closed, but not IO-progressive, is the language of a partial FSM.*

Given a regular language L over alphabet $I \times O$, an algorithm follows to build L^{FSM}, the largest subset of L that is the $\times$-language of an FSM over input alphabet I and output alphabet O.

Procedure 4.1 *Input: Regular Language L over $I \times O$; Output: Largest FSM language L^{FSM} over $I \times O$.*

1. Build a deterministic automaton A accepting L.
2. Delete all nonfinal states together with their incoming edges.
3. If the initial state has been deleted, then $L^{FSM} = \emptyset$. Otherwise, let $\hat{A}$ be the automaton produced by the procedure and L^{FSM} the language that $\hat{A}$ accepts. If there is no outgoing edge from the initial state of $\hat{A}$, then $\hat{A}$ accepts the trivial FSM language $L^{FSM} = \{\epsilon\}$, otherwise it accepts a nontrivial FSM language L^{FSM}. Any FSM language in L must be a subset of L^{FSM}.

To obtain the largest subset of L that is the language of a complete FSM we must apply one more pruning algorithm.

Procedure 4.2 *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 $Prog(L^{FSM})$.

Theorem 4.3 *Procedure 4.2 returns the largest I-progressive subset of L^{FSM}.*

Proof. Define a state s of the automaton A representing L^{FSM} as I_1-nonprogressive if for some $i \in I$ and for all $o \in O$ there is no state reached from s under a transition labeled with (i, o). State s is I_k-nonprogressive, $k > 1$, if for some $i \in I$ and for all $o \in O$ each state reached from s under the transition labeled (i, o) is I_j-nonprogressive, $j < k$. State s is I-nonprogressive if it is I_k-nonprogressive for some $k \geq 1$. The language $Prog(L^{FSM})$ is represented by the automaton $Prog(A)$, obtained from A by removing iteratively the I-nonprogressive states and the related transitions.

We must prove that if $K \subseteq L^{FSM}$ and K is I-progressive then $K \subseteq Prog(L^{FSM})$. The proof goes by induction. If K is I-progressive, there is no string in K that takes the automaton A from the initial state to an I_1-nonprogressive state. Suppose now by induction hypothesis that no string in K takes A to an I_k-nonprogressive state, $k \geq 1$. We must conclude that, if K is I-progressive, there is also no string that takes A to an I_{k+1}-nonprogressive state, otherwise, by definition of I_{k+1}-nonprogressive, K has a string that takes A to some I_j-nonprogressive state, $j \leq k$. Therefore no string in K takes the automaton A to a nonprogressive state, i.e., $K \subseteq Prog(L^{FSM})$. □

Proposition 4.1 *An FSM whose language is L^{FSM} or $Prog(L^{FSM})$ can be deduced trivially from $\hat{A}$ (obtained according to Proc. 4.2) by interpreting each label (i, o) as an input/output pair i/o.*

Proposition 4.2 *Given a regular language L over alphabet $I \times O$, let M be an FSM over input alphabet I and output alphabet O. The language $L_r^{\times}(M)$ of M is contained in L iff $L_r^{\times}(M) \subseteq L^{FSM}$.*

Proof. Show that $L_r^{\times}(M) \subseteq L \Rightarrow L_r^{\times}(M) \subseteq L^{FSM}$. Indeed $L_r^{\times}(M)$ is an FSM language contained in L and L^{FSM} is by construction the largest FSM language contained in L. So $L_r^{\times}(M) \subseteq L^{FSM}$.

$L_r^{\times}(M) \subseteq L^{FSM} \Rightarrow L_r^{\times}(M) \subseteq L$, since by definition $L^{FSM} \subseteq L$. □

Given a regular language L over alphabet $I \cup O$, an algorithm follows to build L^{FSM}, the largest subset of L that is the $\cup$-language of an FSM over input alphabet I and output alphabet O.

Procedure 4.3 *Input: Regular language L over $I \cup O$; Output: Largest FSM language L^{FSM} over $I \cup O$.*

1. Build a deterministic automaton A accepting $L \cap (IO)^*$.
2. Delete the initial state if it is a nonfinal state.
3. Delete all nonfinal states having incoming edges labelled with symbols from alphabet O, together with their incoming edges.
4. If the initial state has been deleted, then $L^{FSM} = \emptyset$. Otherwise, let $\hat{A}$ be the automaton produced by the procedure and L^{FSM} the language that $\hat{A}$ accepts. If there is no outgoing edge from the initial state of $\hat{A}$, then $\hat{A}$ accepts the trivial language $L^{FSM} = \{\epsilon\}$, otherwise it accepts a nontrivial FSM language L^{FSM}. Any FSM language in L must be a subset of L^{FSM}.

To obtain the largest subset of L that is the language of a complete FSM we must apply one more pruning algorithm.

Procedure 4.4 *Input: FSM Language L^{FSM} over $I \cup O$; Output: Largest IO-progressive FSM language $Prog(L^{FSM})$ over $I \cup O$.*

1. Build a deterministic automaton A accepting L^{FSM}.
2. Iteratively delete all states that are final and for which $\exists i \in I$ with no outgoing edge carrying the label i, together with their incoming edges, until the initial state is deleted or no more state can be deleted. Delete the initial state if $\exists i \in I$ with no outgoing edge carrying the label i.

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 IO-progressive FSM language in L^{FSM} must be a subset of $Prog(L^{FSM})$.

Theorem 4.4 *Procedure 4.4 returns the largest IO-progressive subset of L^{FSM}.*

Proof. Similar to the proof of Theorem 4.3. □

Proposition 4.3 *An FSM whose language is L^{FSM} or $Prog(L^{FSM})$ can be deduced trivially from $\hat{A}$ (obtained according to Proc. 4.4) by replacing pairs of consecutive edges labelled, respectively, with i and o by a unique edge labelled i/o.*

Proposition 4.4 *Given a regular language L over alphabet $I \cup O$, let M be an FSM over input alphabet I and output alphabet O. The language $L_r^{\cup}(M)$ of M is contained in L iff $L_r^{\cup}(M) \subseteq L^{FSM}$.*

The proof is the same as the one of Prop. 4.2.

Finally we characterize the Moore FSMs that are the reduction of a given FSM. Notice that the language of a Moore FSM is a Moore language.

Procedure 4.5 *Input: Complete FSM M; Output: Largest submachine of M that is a Moore FSM, denoted by $Moore(M)$, if it exists.*

Given a state $s \in M$, define the set $K_s = \{o \in O \mid \forall i \in I \; \exists s' \in M \; s.t. \; (i, s, s', o) \in T_M\}$, $K_s \subseteq O$.

1. Iterate for each state $s \in M$ until M does not change.
   (a) Compute the set $K_s \subseteq O$.
   (b) If $K_s \neq \emptyset$ delete from T_M each transition (i, s, s', o) such that $o \notin K_s$;
   if $K_s = \emptyset$ delete s with all its incoming edges from M.
2. If the initial state has been deleted then there is no submachine of M that is a Moore FSM, otherwise $Moore(M) = M$.

Theorem 4.5 *Any Moore FSM M' that is a reduction of M is a reduction of* Moore(M), *the output of Proc. 4.5.*

Proof. Define a state s of FSM M as 1-nonMoore if $K_s = \emptyset$. State s is k-nonMoore, $k > 1$, if for some $i \in I$ and for all $o \in O$ each state reached from s under the transition labeled (i/o) is j-nonMoore, $j < k$. State s is nonMoore if it is k-nonMoore for some $k \geq 1$. $Moore(M)$ is obtained from M by removing iteratively the nonMoore states and, from the remaining states s, the transitions (i, s, s', o) such that $o \notin K_s$. Notice that by construction $Moore(M)$ is guaranteed to be complete.

We must prove that if $L(M') \subseteq L(M)$ and $L(M')$ is Moore then $L(M') \subseteq L(Moore(M))$. The proof goes by induction. If $L(M')$ is Moore, there is no string in $L(M')$ that takes the FSM M from the initial state to a 1-nonMoore state. Suppose now by induction hypothesis that no string in $L(M')$ takes M to a k-nonMoore state, $k > 1$. We must conclude that, if $L(M')$ is Moore, there is also no string that takes M to a $(k+1)$-nonMoore state, otherwise, by definition of $(k+1)$-nonMoore, $L(M')$ has a string that takes M to some j-nonMoore state, $j \leq k$. Therefore no string in $L(M')$ takes the FSM M to a nonMoore state, i.e., $L(M') \subseteq L(Moore(M))$. □

Moore machines play a role in guaranteeing that the composition of FSMs is a complete FSM (see Theorem 4.6).

4.3 Composition of Finite State Machines

Different types of composition between pairs of FSMs may be defined, according to the protocol by which signals are exchanged. For a given composition operator and pair of FSMs we must establish whether the composition of this pair is defined, meaning that it yields a set of behaviours that can be described by another FSM. In general, the composition of FSMs is a partially specified function from pairs of FSMs to an FSM. In our approach we define the composition of FSMs by means of the composition operators over languages introduced in Sec. 2. Thus the FSM yielded by the composition of FSMs M_A and M_B is the one whose language is obtained by the composition of the FSM languages associated to M_A and M_B. The synchronous composition operator models the synchronous connection of sequential circuits, while the parallel composition operator models an exchange protocol by which an input is followed by an output after a finite exchange of internal signals. The latter model, introduced in [29], abstracts a system with two components and a single message in transit. At any moment either the components exchange messages or one of them communicates with its environment. The environment submits the next external input to the system only after the system has produced an external output in response to the previous input.

4.3.1 Synchronous Composition of FSMs

Consider the pair of FSMs

1. FSM M_A has input alphabet $I_1 \times V$, output alphabet $U \times O_1$ and transition relation T_A;
2. FSM M_B has input alphabet $I_2 \times U$, output alphabet $V \times O_2$ and transition relation T_B.

We define a synchronous composition operator $\bullet$ that associates to a pair of FSMs M_A and M_B another FSM $M_A \bullet M_B$ such that

1. the external input alphabet is $I_1 \times I_2 = I$;
2. the external output alphabet is $O_1 \times O_2 = O$.

Recall that, by definition of synchronous composition of languages, a sequence $\alpha \in (I_1 \times I_2 \times O_1 \times O_2)^*$ is in the language of the synchronous composition of $L(M_A)$ and $L(M_B)$ iff α is in the projection onto $I_1 \times I_2 \times O_1 \times O_2$ of the intersection of the liftings, respectively, of $L(M_A)$ over $I_2 \times O_2$ and of $L(M_B)$ over $I_1 \times O_1$ [4]:

$$\alpha \in L(M_A) \bullet L(M_B) \text{ iff}$$

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

Notice that the liftings $L(M_A)_{\uparrow I_2 \times O_2}$ and $L(M_B)_{\uparrow I_1 \times O_1}$ are needed to have the languages of M_A and M_B defined on the same alphabet; e.g., $L(M_B)$ is defined over $I_2 \times U \times V \times O_2$, and the lifting $\uparrow I_1 \times O_1$ defines it over $I_1 \times I_2 \times U \times V \times O_1 \times O_2$.

Lemma 4.1 *If $L(M_A)$ and $L(M_B)$ are FSM $\times$-languages, then $L(M_A) \bullet L(M_B)$ is an FSM $\times$-language.*

Proof. $L(M_A) \bullet L(M_B)$ is prefix-closed, because prefix-closed FSM $\times$-languages are closed under $\bullet$ composition. Notice that $L(M_A) \bullet L(M_B)$ does not need to be progressive, because partial FSMs are allowed. □

Therefore we can state the following definition.

Definition 4.10 *The* **synchronous composition of FSMs** *M_A and M_B yields the FSM $M_A \bullet M_B$ with language*

$$L(M_A \bullet M_B) = L(M_A) \bullet L(M_B).$$

If the language $L(M_A) \bullet L(M_B) = \{\epsilon\}$, then $M_A \bullet M_B$ is a trivial FSM.

[4] Use the same order $I_1 \times I_2 \times U \times V \times O_1 \times O_2$ in the languages $L(M_A)_{\uparrow I_2 \times O_2}$ and $L(M_B)_{\uparrow I_1 \times O_1}$.

The previous definition is sound because the language $L(M_A) \bullet L(M_B)$ by Lemma 4.1 is an FSM language, which may correspond to a complete or partial FSM according to whether the language $L(M_A) \bullet L(M_B)$ is I-progressive or not. Then by subset construction and reduction we produce a reduced observable FSM. In summary, 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 an FSM; then we produce a reduced PNDFSM (we assume that M_A and M_B are PNDFSMs), because subset construction determinizes the underlying finite automaton.

Example 4.1 *a) Synchronous composition of two FSMs defining a complete FSM.*

Consider the FSMs $M_A = \langle S_A, I_1 \times V, U \times O_1, T_A, s1\rangle$ *and* $M_B = \langle S_B, U, V, T_B, sa\rangle$ *with* $S_A = \{s1, s2, s3\}$, $T_A = \{(1\text{ -}, s1, s1, 11), (00. s1, s2, 10), (01, s1, s3, 10), (-\ 0, s2, s1, 01), (-\ 1, s2, s3, 10), (-\ 1, s3, s1, 01), (-\ 0, s3, s2, 00)\}$, $S_B = \{sa, sb\}$, $T_B = \{(0, sa, sa, 1), (1\ sa\ sb\ 0), (0, sb, sa, 0), (1, sb, sb, 0)\}$ [5].

Then $M_A \bullet M_B = M_{A\bullet B} = \langle S_{A\bullet B}, I_1, O_1, T_{A\bullet B}, (s1, sa)\rangle$ *with* $S_{A\bullet B} = \{(s1, sa), (s1, sb), (s2, sb)\}$ *and* $T_{A\bullet B} = \{(1, (s1, sa), (s1, sb), 1), (0, (s1, sa), (s2, sb), 0), (1, (s1, sb), (s1, sb), 1)\ (0, (s1, sb), (s2, sb), 0), (-, (s2, sb), (s1, sa), 1)\}$ *is a complete FSM.*

b) Synchronous composition of two FSMs defining a partial FSM.

Modify the transition relation of M_B *as follows:* $T_B = \{(0, sa, sa, 1), (1\ sa\ sb\ 0), (0, sb, sa, 1), (1, sb, sb, 0)\}$. *Then* $T_{A\bullet B} = \{(1, (s1, sa), (s1, sb), 1), (0, (s1, sa), (s2, sb), 0), (1, (s1, sb), (s1, sb), 1)\ (0, (s1, sb), (s2, sb), 0)\}$ *defines a partial FSM (no transition from state* $(s2, sb)$*).*

Theorem 4.6 *Let* M_A *be a complete FSM over input alphabet* $I_1 \times V$ *and output alphabet* $O_1 \times U$ *and let* M_B *be a complete Moore FSM over input alphabet* $I_2 \times U$ *and output alphabet* $O_2 \times V$. *Then the composition* $M_A \bullet M_B$ *is a complete FSM.*

Proof. Consider a string $\alpha \in L(M_A)_{\uparrow I_2 \times O_2} \cap L(M_B)_{\uparrow I_1 \times O_1}$. Suppose that from the initial state M_A reaches state s under the string $\alpha_{\downarrow I_1 \times U \times V \times O_1}$ and similarly that M_B reaches state t under the string $\alpha_{\downarrow I_2 \times U \times V \times O_2}$. Let the external input $(i_1, i_2) \in I_1 \times I_2$ be applied next. For any $u \in U$ there is a transition $i_2u, t, t', o_2v)$ in M_B, because M_B is a complete FSM; similarly, for any $v \in V$ there is a transition (i_1v, s, s', o_1u') in M_A, because M_A is a complete FSM. Moreover, given the input i_2u', there is a transition (i_2u', t, t'', o_2v) with the same output o_2v of transition (i_2u, t, t', o_2v), because M_B is a Moore FSM. Therefore u' and v are matching internal signals, i.e., the string α can be extended by $(i_1, i_2, u', v, o_1, o_2)$. □

4.3.2 Parallel Composition of FSMs

Consider the pair of FSMs [6]

1. FSM M_A has input alphabet $I_1 \cup V$, output alphabet $U \cup O_1$ and transition relation T_A;
2. FSM M_B has input alphabet $I_2 \cup U$, output alphabet $V \cup O_2$ and transition relation T_B.

We define a parallel composition operator $\diamond$ that associates to a pair of FSMs M_A and M_B another FSM $M_A \diamond M_B$ such that:

1. the alphabet of the external inputs is $I_1 \cup I_2 = I$;
2. the alphabet of the external outputs is $O_1 \cup O_2 = O$.

[5] – denotes input or output don't care conditions.

[6] For simplicity the alphabets I_1, I_2, O_1, O_2, U, V are assumed to be disjoint.

Recall that, by definition of parallel composition of languages, a sequence $\alpha \in ((I_1 \cup I_2)(O_1 \cup O_2))^*$ is in the language of the parallel composition of $L(M_A)$ and $L(M_B)$ iff α is in the restriction onto $I_1 \cup I_2 \cup O_1 \cup O_2$ of the intersection of the expansions, respectively, of $L(M_A)$ over $I_2 \cup O_2$ and of $L(M_B)$ over $I_1 \cup O_1$:

$$\alpha \in L(M_A) \diamond L(M_B) \text{ iff}$$

$$\alpha \in [L(M_A)_{\Uparrow I_2 \cup O_2} L \cap L(M_B)_{\Uparrow I_1 \cup O_1}]_{\Downarrow I \cup O}.$$

Notice that the expansions $L(M_A)_{\Uparrow I_2 \cup O_2}$ and $L(M_B)_{\Uparrow I_1 \cup O_1}$ are needed to have the languages of M_A and M_B defined on the same alphabet; e.g., $L(M_B)$ is defined over $I_2 \cup U \cup V \cup O_2$, and the expansion $\Uparrow I_1 \times O_1$ defines it over $I_1 \cup I_2 \cup U \cup V \cup O_1 \cup O_2$.

Lemma 4.2 *If $L(M_A)$ and $L(M_B)$ are FSM $\cup$-languages, then $L(M_A) \diamond L(M_B) \cap (IO)^*$ is an FSM $\cup$-language.*

Proof. $L(M_A) \diamond L(M_B) \cap (IO)^*$ is IO-prefix-closed, because IO-prefix-closed $\cup$-languages are closed under $\diamond$ composition. Indeed, a state of the finite automaton corresponding to an FSM $\cup$-language is accepting iff it is the initial state or all its ingoing edges are labelled by symbols in O. The property is preserved by intersection and restriction over $I \cup O$. The intersection with $(IO)^*$ makes sure that in the strings of the resulting FSM $\cup$-language an input is always followed by exactly one output, so that a corresponding FSM (with edges labelled by pairs (i/o)) can be reconstructed. Notice that $L(M_A) \diamond L(M_B) \cap (IO)^*$ does not need to be IO-progressive, because partial FSMs are allowed. □

Therefore we can state the following definition.

Definition 4.11 *The* **parallel composition of FSMs** *M_A and M_B yields the FSM $M_A \diamond M_B$ with language*

$$L(M_A \diamond M_B) = L(M_A) \diamond L(M_B) \cap (IO)^*.$$

If the language $L(M_A) \diamond L(M_B) \cap (IO)^* = \{\epsilon\}$, then $M_A \diamond M_B$ is a trivial FSM.

The previous definition is sound because the language $L(M_A) \diamond L(M_B) \cap (IO)^*$ by Lemma 4.2 is an FSM language, which may correspond to a complete or partial FSM according to whether the language $L(M_A) \diamond L(M_B) \cap (IO)^*$ is IO-progressive or not. Then by subset construction and reduction we produce a reduced observable FSM.

Example 4.2 *a) Parallel composition of two FSMs defining a complete FSM.*

Consider the FSMs $M_A = \langle S_A, I_1 \cup V, U \cup O_1, T_A, s1 \rangle$ and $M_B = \langle S_B, U, V, T_B, sa \rangle$ with $S_A = \{sa, sb\}$, $T_A = \{(i_1, sa, sa, o_1), (v_1, sa, sa, o_1), (v_2, sa, sa, u_1), (i_2, sa, sb, u_2), (v_1, sb, sb, u_1), (v_2, sb, sb, o_2), (i_1, sb, sa, o_1), (i_2, sb, sa, o_1)\}$, $S_B = \{s1, s2\}$, $T_B = \{(u_2, s1, s2, v_2), (u_1\, s1, s2, v_1), (u_1, s2, s2, v_2), (u_2, s2, s1, v_2)\}$.

Then the composition $M_A \diamond M_B = M_{A \diamond B} = \langle S_{A \diamond B}, I_1, O_1, T_{A \diamond B}, (sa, s1) \rangle$ with $S_{A \diamond B} = \{(sa, s1), (sb, s2), (sa, s2), (sb, s1)\}$ and $T_{A \diamond B} = \{(i_1, (sa, s1), (sa, s1), o_1), (i_2, (sa, s1), (sb, s2), o_2), (i_1, (sb, s2), (sa, s2), o_1), (i_2, (sb, s2), (sa, s2), o_1), (i_1, (sa, s2), (sa, s2), o_1), (i_2, (sa, s2), (sb, s1), o_2), (i_1, (sb, s1), (sa, s1), o_1), (i_2, (sb, s1), (sa, s1), o_1)\}$ is a complete FSM.

Fig. 3 shows some steps and the result of the computation of $M_A \diamond M_B$.

b) Parallel composition of two FSMs defining a partial FSM.

Modify the transition relation of M_B as follows: $T_B = \{(u_2, s1, s2, v_2), (u_1\, s1, s2, v_1), (u_1, s2, s2, v_2), (u_2, s2, s1, v_1)\}$. Then $T_{A \diamond B} = \{(i_1, (sa, s1), (sa, s1), o_1), (i_2, (sa, s1), (sb, s2), o_2), (i_1, (sb, s2), (sa, s2), o_1), (i_2, (sb, s2), (sa, s2), o_1), (i_1, (sa, s2), (sa, s2), o_1)\}$ defines an incomplete FSM (no transition from state $(sa, s2)$ under input i_2).

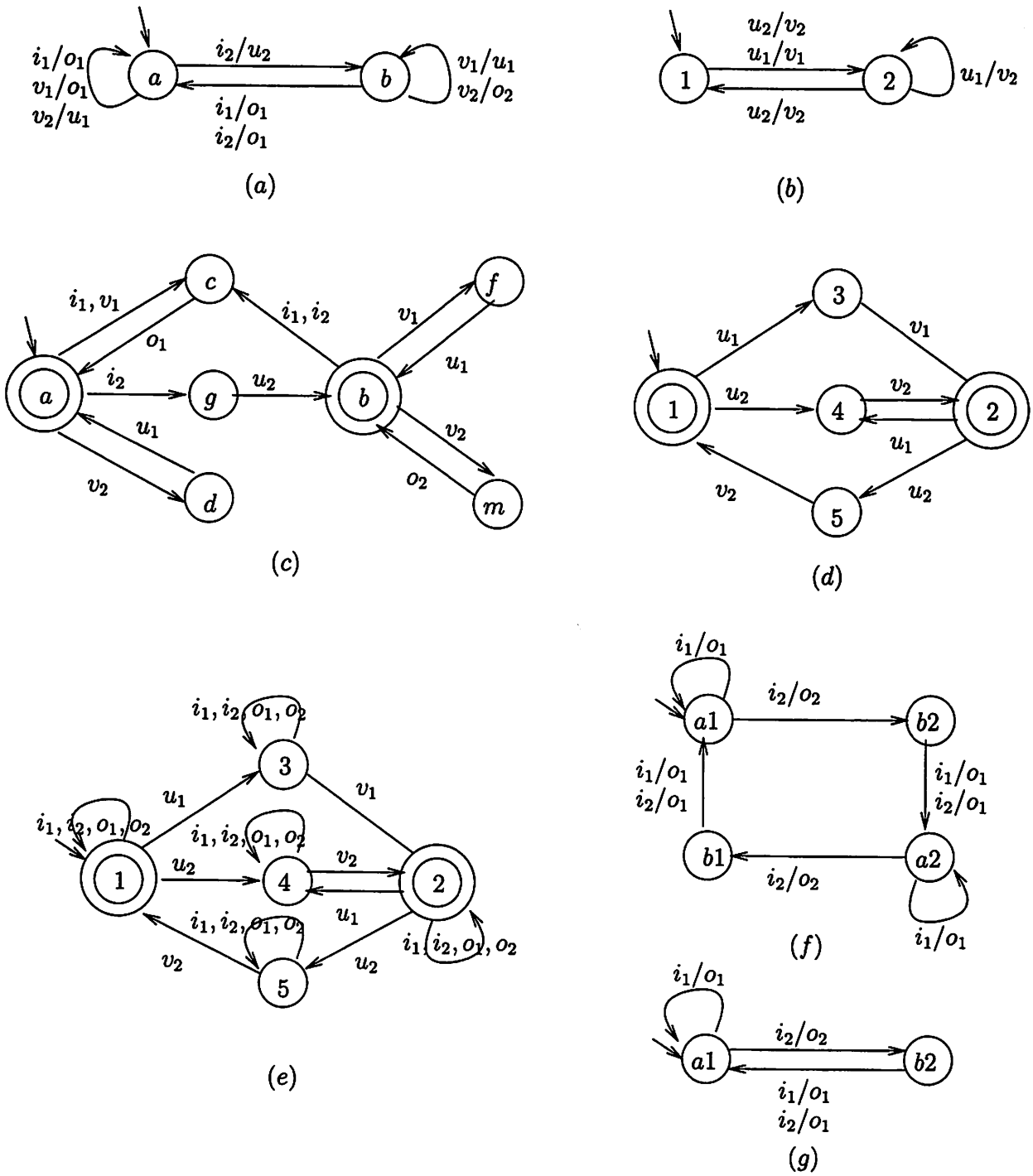

Figure 3: Illustration of parallel composition $M_A \diamond M_B = M_{A \diamond B}$ of Example 4.2-a). (a) FSM M_A; (b) FSM M_B; (c) Automaton of A (∪-language of M_A); (d) Automaton of B (∪-language of M_B); (e) Automaton of $B_{\Uparrow I_1 \cup O_1}$; (f) FSM $M_A \diamond M_B = M_{A \diamond B}$; (g) Reduced FSM $M_A \diamond M_B = M_{A \diamond B}$.

4.4 Definition of Equations over FSMs

Consider the network of FSMs shown in Fig. 4, where FSM M_A has input signals I_1 and V and output signals U and O_1, and FSM M_B has input signals I_2 and U and output signals V and O_2. The network implements a specification M_C with input signals I_1, I_2 and output signals O_1, O_2. Supposing that M_A and M_C are known and M_B is unknown, we want to define an equation of the type $M_A \odot M_X \approx M_C$, to capture the FSMs M_B that in place of M_X let the network of M_A and M_B match the specification M_C. Through Definitions 4.6 and 4.7 we have seen two different ways to associate an FSM language to a given

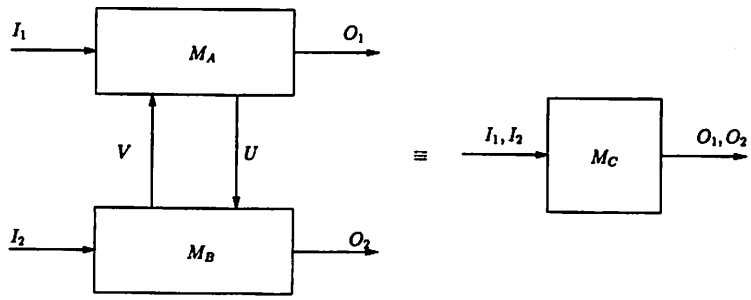

Figure 4: General Topology.

FSM, and related composition operators $\bullet$ and $\diamond$ have been introduced in Sec. 4.3; therefore we introduce two types of equations over FSMs:

$$M_A \bullet M_X \preceq M_C$$

and

$$M_A \diamond M_X \preceq M_C,$$

and solve them by building first the related language equations

$$L(M_A) \bullet L(M_X) \subseteq L(M_C)$$

and

$$L(M_A) \diamond L(M_X) \subseteq L(M_C) \cup \overline{(IO)^*},$$

where $L(M_A)$ and $L(M_C)$ are the FSM languages associated with FSMs M_A and M_C. The latter language equation is justified by the following chain of equivalences

$$\begin{aligned} M_A \diamond M_X \preceq M_C &\Leftrightarrow \\ L(M_A \diamond M_X) \subseteq L(M_C) &\Leftrightarrow \textit{ by Def. 4.11} \\ L(M_A) \diamond L(M_X) \cap (IO)^* \subseteq L(M_C) &\Leftrightarrow \\ L(M_A) \diamond L(M_X) \subseteq L(M_C) \cup \overline{(IO)^*}. & \end{aligned}$$

The last equivalence uses the set-theoretic equality $AB \subseteq C \Leftrightarrow A \subseteq C + \overline{B}$ [7].

When there is no ambiguity we will denote by $A \bullet X \subseteq C$ and $A \diamond X \subseteq C \cup \overline{(IO)^*}$ the language equations $L(M_A) \bullet L(M_X) \subseteq L(M_C)$ and $L(M_A) \diamond L(M_X) \subseteq L(M_C) \cup \overline{(IO)^*}$, where $L(M_A)$, $L(M_X)$ and $L(M_C)$ are, respectively, the $\times$-languages and $\cup$-languages associated with the FSMs M_A, M_X and M_C.

[7] In one direction, $AB \subseteq C \Rightarrow AB + A\overline{B} \subseteq C + A\overline{B} \Rightarrow A \subseteq C + A\overline{B} \Rightarrow A \subseteq C + \overline{B}$. In the other direction, $A \subseteq C + \overline{B} \Rightarrow AB + A\overline{B} \subseteq C + \overline{B} \Rightarrow AB \subseteq C$, because $AB \not\subseteq \overline{B}$.

4.5 FSM Equations under Synchronous Composition

4.5.1 Largest FSM Solutions

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 equation

$$M_A \bullet M_X \preceq M_C, \tag{5}$$

whose unknown is an FSM M_X over inputs $I_2 \times U$ and outputs $V \times O_2$. Sometimes the shortened notation $I = I_1 \times I_2$ and $O = O_1 \times O_2$ will be used.

Definition 4.12 *FSM M_B is a* **solution** *of the equation $M_A \bullet M_X \preceq M_C$, where M_A and M_C are FSMs, iff $M_A \bullet M_B \preceq M_C$.*

Converting to the related FSM languages, we construct the associated language equation

$$L(M_A) \bullet L(M_X) \subseteq L(M_C), \tag{6}$$

where $L(M_A)$ is an FSM language over alphabet $I_1 \times U \times V \times O_1$, $L(M_C)$ is an FSM language over alphabet $I_1 \times I_2 \times O_1 \times O_2$ and the unknown FSM language is over alphabet $I_2 \times U \times V \times O_2$. The previous equation can be rewritten for simplicity as

$$A \bullet X \subseteq C. \tag{7}$$

We want to characterize the solutions of $A \bullet X \subseteq C$ as FSM languages. We know from Theorem 2.1 that the largest solution of the equation $A \bullet X \subseteq C$ is the language $S = \overline{A \bullet \overline{C}}$.

When A and C are FSM languages, the following property holds.

Theorem 4.7 $S \neq \emptyset$.

Proof. FSM languages are not closed under complementation, because the complement of an FSM language does not include the empty string ϵ, so it cannot be an FSM language. So $\overline{C}$ does not include ϵ and neither does $\overline{C}_{\uparrow U \times V}$. Then the intersection $A_{\uparrow I_2 \times O_2} \cap \overline{C}_{\uparrow U \times V}$ does not include the empty string, neither does its projection $(A_{\uparrow I_2 \times O_2} \cap \overline{C}_{\uparrow U \times V})_{\downarrow I_2 \times U \times V \times O_2}$, that is $A \bullet \overline{C}$. Therefore $\overline{A \bullet \overline{C}}$ includes the empty string, i.e., $\epsilon \in S \neq \emptyset$. □

Example 4.3 *Consider the FSMs $M_A = \langle S_A, I_1 \times V, U \times O_1, T_A, sa \rangle$ and $M_C = \langle S_C, U, V, T_C, s1 \rangle$ with $S_A = \{sa\}$, $T_A = \{(01, sa, sa, 01), (00, sa, sa, 01), (11, sa, sa, 10), (10, sa, sa, 10)\}$, $S_C = \{s1\}$, $T_C = \{(1, s1, s1, 1), (0, s1, s1, 0)\}$. The equation $M_A \bullet M_X \preceq M_C$ yields the language equation $A \bullet X \subseteq C$ with solution $S = \{\epsilon\}$, i.e., the corresponding FSM solution M_X produces only the empty word.*

It is also true that if M_C produces only ϵ, then M_X produces only ϵ.

In general S is not an FSM language. To compute the largest FSM language contained in S, that is S^{FSM}, we must compute the largest prefix-closed language contained in S.

Theorem 4.8 *Let A and C be FSM languages. The largest FSM language that is a solution of the equation $A \bullet X \subseteq C$ is given by S^{FSM}, where $S = \overline{A \bullet \overline{C}}$. S^{FSM} is obtained by applying Procedure 4.1 to S. S^{FSM} contains at least the string ϵ.*

Thus, a synchronous FSM language equation is always solvable, since the solution includes at least the empty string and so its prefix-closure does too. This corresponds to the trivial FSM being always a solution of a synchronous FSM equation.

Example 4.4 *Consider the equation $M_A \bullet M_X \preceq M_C$, with M_A and M_C shown, respectively, in Fig. 5(a) and 5(c). The automata of the related languages are shown, respectively, in Fig. 5(b) and 5(d). The intermediate steps to compute the solution are demonstrated in Fig. 5(e)-(g). The automaton generating the largest language solution, $S = \overline{(A \cap (\overline{C}_{\uparrow U \cup V})_{\downarrow U \cup V}}$, is portrayed in Fig. 5(h). Notice that it is not prefix-closed, since string $u_1 v_1 u_2 v_1 \in S$, but $u_1 v_1 \notin S$; its largest prefix-closed sublanguage yields the largest FSM solution M_X shown in Fig. 5(i). The composition of any DFSM reduction of M_X with M_A produces the trivial machine.*

For logic synthesis applications, we assume that M_A and M_C are complete FSMs and we require that the solution is a complete FSM too. This is obtained by applying Procedure 4.2 to S^{FSM}, yielding $Prog(S^{FSM})$, the largest $(I_2 \times U)$-progressive FSM language $\subseteq (I_2 \times U \times V \times O)^*$. Notice that an $(I_2 \times U)$-progressive solution might not exist, and in that case Procedure 4.2 returns an empty language.

Proposition 4.5 *FSM M_B is a solution of the equation $M_A \bullet M_X \preceq M_C$, where M_A and M_C are FSMs, iff M_B is a reduction of the FSM M_S associated to S^{FSM}, where S^{FSM} is obtained by applying Procedure 4.1 to S, where $S = \overline{A \bullet \overline{C}}$. If $S^{FSM} = \{\epsilon\}$ then the trivial FSM is the only solution. The largest complete FSM solution $M_{Prog(S^{FSM})}$ is found, if it exists, by applying Procedure 4.2. A complete FSM is a solution iff it is a reduction of the largest complete solution $M_{Prog(S^{FSM})}$.*

Example 4.5 *(Variant of Ex. 4.3) Consider the FSMs $M_A = \langle S_A, I_1 \times V, U \times O_1, T_A, sa \rangle$ and $M_C = \langle S_C, U, V, T_C, s1 \rangle$ with $S_A = \{sa\}$, $T_A = \{(01,\ sa,\ sa,\ 01),\ (00,\ sa,\ sa,\ 01),\ (11,\ sa,\ sa,\ 10),\ (10,\ sa,\ sa,\ 10)\}$, $S_C = \{s1\}$, $T_C = \{(1,\ s1,\ s1,\ 1),\ (0,\ s1,\ s1,\ 0)\}$. The equation $M_A \bullet M_X \preceq M_C$ yields the language equation $A \bullet X \subseteq C$ with solution $S = \{(01)^*\}$, i.e., the corresponding FSM solution M_X produces the set of strings of input/output pairs $\{(0/1)^*\}$, and so the equation has no complete FSM solution.*

4.5.2 Computational Complexity

Consider the equation $A \bullet X \subseteq C$, where A and C are FSM languages. We know that the largest unconstrained solution is given by $S = \overline{A \bullet \overline{C}}$. Given the rectification topology in Fig. 1(d) with M_B as the unknown M_X, the solution has in the worst-case $2^{|S_A|.2^{|S_C|}}$ states, where S_A is the number of states of FA A and S_C is the number of states of FA C. The exponent $2^{|S_C|}$ appears when C is non-deterministic, to account for the determinization needed to compute $\overline{C}$; otherwise if C is deterministic, in place of $2^{|S_C|}$ we have $|S_C|$, because complementation becomes a linear operation. The product $|S_A|.2^{|S_C|}$ is due to the product of automata A and C. Then to complete the computation of the $\bullet$ operator we must project the product on the internal signals U and V; as a result we may get again a non-deterministic automaton, and therefore a new determinization is needed before performing the final complementation: this explains the outer exponential $2^{|S_A|.2^{|S_C|}}$.

There are "easier" topologies, like supervisory control, where there is no projection onto a subset of signals of the product automaton; therefore non-determinism is not introduced and so the final complementation is linear, resulting in the complexity of $|S_A|.2^{|S_C|}$. Moreover, if S_C is deterministic, the exponential $2^{|S_C|}$ is replaced by $|S_C|$, and so the final complexity of supervisory control is bounded by only $|S_A|.|S_C|$.

The operations on the language S to extract from it an FSM language, a complete FSM language or a Moore solution (see below) are linear in the number of edges of the automaton representing S.

4.5.3 Largest FSM Compositional Solutions

It is interesting to compute the subset of compositionally I-progressive solutions B, i.e., such that $A_{\uparrow I_2 \times O_2} \cap B_{\uparrow I_1 \times O_1}$ is an I-progressive FSM language $\subseteq (I \times U \times V \times O)^*$. Thus the composition (after projection on the external signals) is the language of a complete FSM over inputs $I_1 \times I_2$ and outputs $O_1 \times O_2$. Since $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{\uparrow I_1 \times O_1}$ is prefix-closed and hence corresponds to a partial FSM, we have to restrict it so that it is also I-progressive, which corresponds to a complete FSM. If S^{FSM} is compositionally I-progressive, then

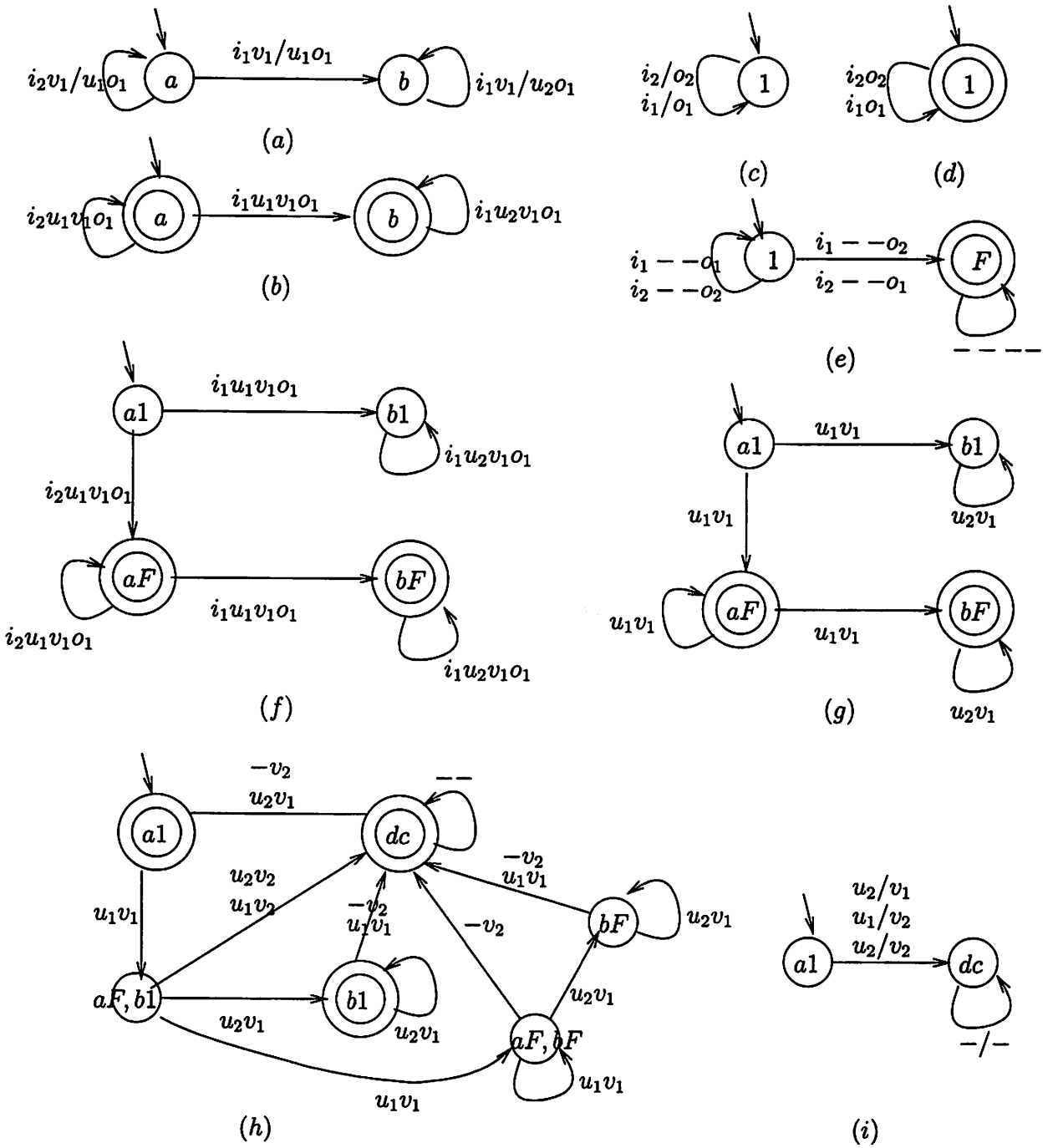

Figure 5: Illustration of Example 4.4. (a) FSM M_A; (b) FA (finite automaton) of $A = L_r^{\times}(M_A)$; (c) FSM M_C; (d) FA of $C = L_r^{\times}(M_C)$; (e) FA of $\overline{C}_{\uparrow U\times V}$; (f) FA of $A \cap \overline{C}_{\uparrow U\times V}$; (g) FA of $(A \cap \overline{C}_{\uparrow U\times V})_{\downarrow U\times V}$; (h) FA of largest solution $S = \overline{(A \cap \overline{C}_{\uparrow U\times V})_{\downarrow U\times V}}$; (i) Largest FSM solution M_X.

S^{FSM} is the largest compositionally I-progressive solution of the equation. However, not every non-empty subset of S^{FSM} inherits the feature of being compositionally I-progressive. If S^{FSM} is not compositionally I-progressive, then denote by $cProg(S^{FSM})$ the largest compositionally I-progressive subset of S^{FSM}. Conceptually $cProg(S^{FSM})$ is obtained from S^{FSM} by deleting each string α such that, for some $i \in I$, there is no $(u, v, o) \in U \times V \times O$ for which it holds $\alpha\ (i, u, v, o) \in A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{\uparrow I_1 \times O_1}$. The following procedure tells how to compute $cProg(S^{FSM})$.

Procedure 4.6 *Input: Largest prefix-closed solution S^{FSM} of synchronous equation $A \bullet X \subseteq C$ and context A; Output: Largest compositionally I-progressive prefix-closed solution $cProg(S^{FSM})$.*

1. Initialize i to 1 and S^i to S^{FSM}.
2. Compute $R^i = A_{\uparrow I_2 \times O_2} \cap S^i_{\uparrow I_1 \times O_1}$.
   If the language R^i is I-progressive then $cProg(S^{FSM}) = S^i$.
   Otherwise
   (a) Obtain $Prog(R^i)$, the largest I-progressive subset of R^i, by using Proc. 4.2.
   (b) Compute $T^i = S^i \setminus (R^i \setminus Prog(R^i))_{\downarrow I_2 \times U \times V \times O_2}$.
3. If $T^{i\ FSM} = \emptyset$ then $cProg(S^{FSM}) = \emptyset$.
   Otherwise
   (a) Assign the language $T^{i\ FSM}$ to S^{i+1}.
   (b) Increment i by 1 and go to 2.

Theorem 4.9 *Proc. 4.6 returns the largest compositionally I-progressive (prefix-closed) solution, if it terminates.*

Theorem 4.10 *Proc. 4.6 terminates.*

The proofs can be found in [40].

A sufficient condition to insure that $A_{\uparrow I_2 \times O_2} \cap S^{FSM}_{\uparrow I_1 \times O_1}$ is an I-progressive FSM language is that $S^{FSM}_{\uparrow I_1 \times O_1}$ or $A_{\uparrow I_2 \times O_2}$ satisfy the Moore property (see Theorem 4.6 for a related statement proved for complete FSMs). If S^{FSM} is Moore then it is the largest Moore solution of the equation. However, not every non-empty subset of S^{FSM} inherits the feature of being Moore. If S^{FSM} is not Moore, then denote by $Moore(S^{FSM})$ the largest Moore subset of S^{FSM}. The set $Moore(S^{FSM})$ is obtained by deleting from S^{FSM} each string which causes S^{FSM} to fail the Moore property.

Proposition 4.6 *If $Moore(S^{FSM}) \neq \emptyset$, then it is the largest Moore solution of the equation $A \bullet X \subseteq C$. Otherwise the equation $A \bullet X \subseteq C$ has no Moore solution.*

To compute the largest Moore FSM that is a solution, it is sufficient to apply Procedure 4.5 to the FSM $M_{S^{FSM}}$ associated to S^{FSM}, as justified by Theorem 4.5. The result is the largest Moore FSM solution, of which every deterministic Moore solution is a reduction.

Example 4.6 *Consider the equation $M_A \bullet M_X \preceq M_C$, with M_A and M_C shown, respectively, in Fig. 6(a) and 6(b). The largest FSM solution M_X is shown in Fig. 6(c), whereas Fig. 6(d) shows the largest Moore FSM solution* Moore(M_X). *Moore FSM solutions are portrayed in Fig. 7(a)-(b), whereas nonMoore FSM solutions are pictured in Fig. 7(c)-(d).*

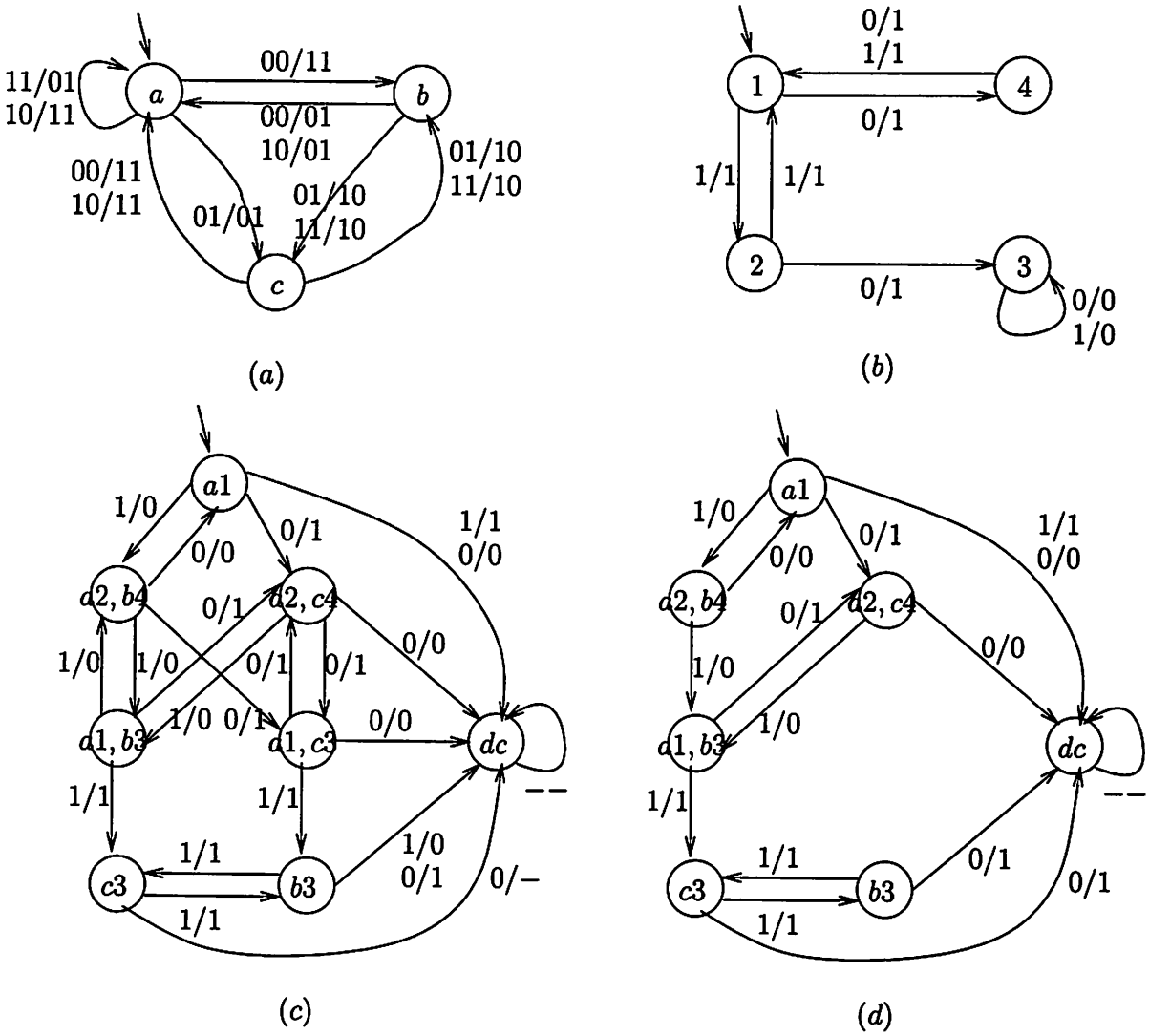

Figure 6: Illustration of Example 4.6. (a) FSM M_A; (b) FSM M_C; (c) Largest FSM solution M_X; (d) Largest Moore FSM solution $Moore(M_X)$.

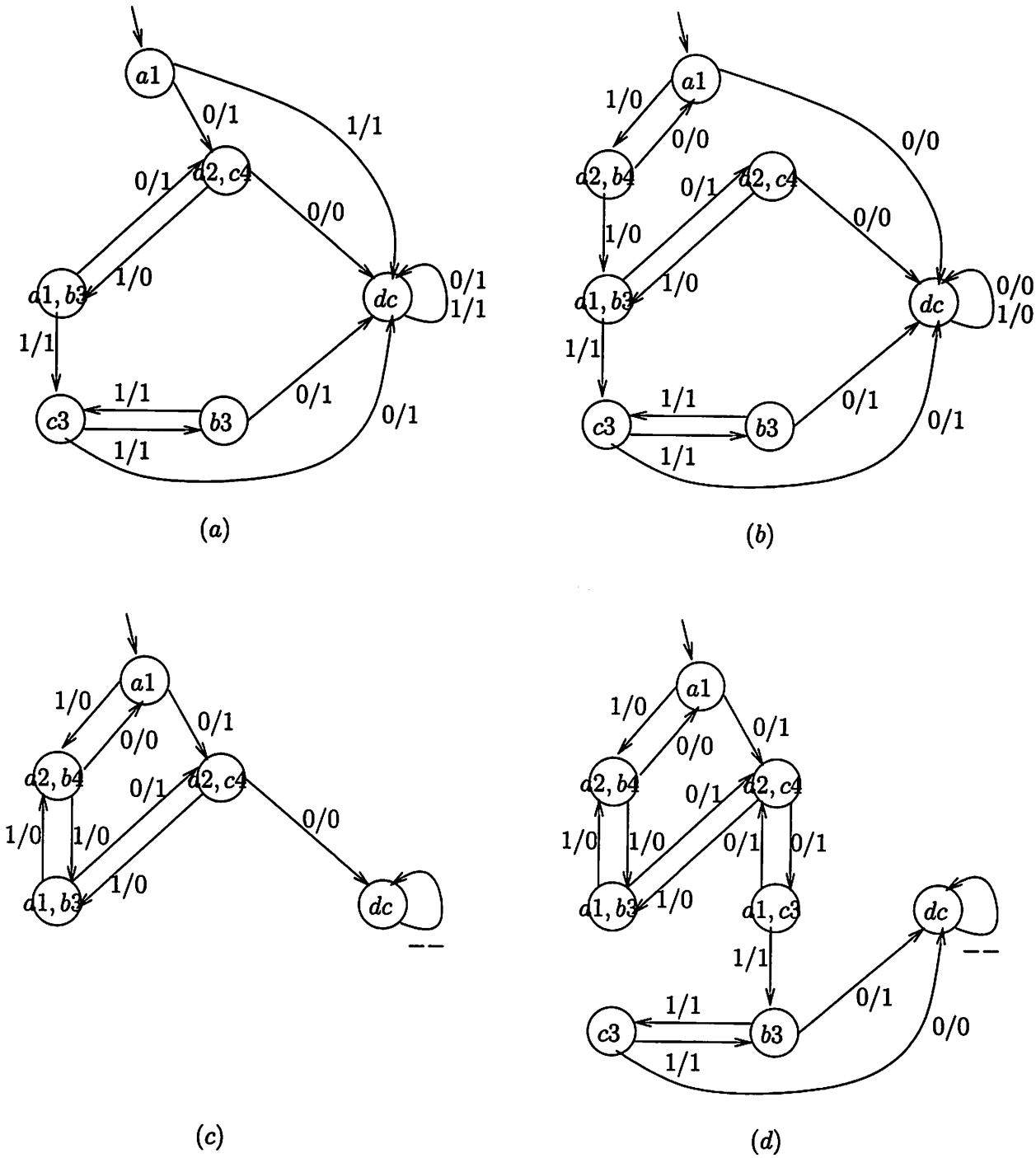

Figure 7: Illustration of Example 4.6. (a)-(b) Moore FSM solutions; (c)-(d) Non-Moore FSM solutions.

4.6 FSM Equations under Parallel Composition

4.6.1 Largest FSM Solutions

Given alphabets I_1, I_2, U, V, O_1, O_2, an FSM M_A over inputs $I_1 \cup V$ and outputs $U \cup O_1$, and an FSM M_C over inputs $I_1 \cup I_2$ and outputs $O_1 \cup O_2$, consider the FSM equation

$$M_A \diamond M_X \preceq M_C, \tag{8}$$

whose unknown is an FSM M_X over inputs $I_2 \cup U$ and outputs $V \cup O_2$. Sometimes the shortened notation $I = I_1 \cup I_2$ and $O = O_1 \cup O_2$ will be used.

Definition 4.13 *FSM M_B is a* **solution** *of the equation $M_A \diamond M_X \preceq M_C$, where M_A and M_C are FSMs, iff $M_A \diamond M_B \preceq M_C$.*

Converting to the related FSM languages, we construct the associated language equation (see Sec. 4.4)

$$L(M_A) \diamond L(M_X) \subseteq L(M_C) \cup \overline{(IO)^*}, \tag{9}$$

where $L(M_A)$ is an FSM language over alphabet $I_1 \cup U \cup V \cup O_1$, $L(M_C)$ is an FSM language over alphabet $I_1 \cup I_2 \cup O_1 \cup O_2$ and the unknown FSM language is over alphabet $I_2 \cup U \cup V \cup O_2$. The previous equation can be rewritten for simplicity as

$$A \diamond X \subseteq C \cup \overline{(IO)^*}. \tag{10}$$

We want to characterize the solutions of $A \diamond X \subseteq C \cup \overline{(IO)^*}$ that are FSM languages. We know from Theorem 2.2 that the largest solution of the equation $A \diamond X \subseteq C \cup \overline{(IO)^*}$ is the language $S = \overline{A \diamond (\overline{C} \cap (IO)^*)}$.

In general S is not an FSM language. To compute the largest FSM language contained in S, that is S^{FSM}, we must compute the largest prefix-closed language contained in $S \cap ((I_2 \cup U)(V \cup O_2))^*$.

Theorem 4.11 *Let A and C be FSM languages. The largest FSM language that is a solution of the equation $A \diamond X \subseteq C \cup \overline{(IO)^*}$ is given by S^{FSM}, where $S = \overline{A \diamond (\overline{C} \cap (IO)^*)}$. If $S = \emptyset$ then $S^{FSM} = \emptyset$; if $S \neq \emptyset$, S^{FSM} is obtained by applying Procedure 4.3 to S. If $S^{FSM} = \emptyset$ then the FSM language equation $A \diamond X \subseteq C \cup \overline{(IO)^*}$ has no solution.*

Proof. The first step of Procedure 4.3 computes the intersection of S with $((I_2 \cup U)(V \cup O_2))^*$ to enforce that the solution, if it exists, is an FSM language with input alphabet $I_2 \cup U$ and output alphabet $V \cup O_2$. Since A and C are regular languages, $S \cap ((I_2 \cup U)(V \cup O_2))^*$ is a regular language too and, by construction, Procedure 4.3 extracts the largest FSM language contained in it. □

Example 4.7 *Consider the FSMs $M_A = \langle S_A, I_1 \cup V, U \cup O_1, T_A, sa \rangle$ and $M_C = \langle S_C, U, V, T_C, s1 \rangle$ with $S_A = \{sa\}$, $T_A = \{(i,\ sa,\ sa,\ o_2),\ (v,\ sa,\ sa,\ u)\}$, $S_C = \{s1\}$, $T_C = \{(i,\ s1,\ s1,\ o_1)\}$. The equation $M_A \diamond M_X \preceq M_C$ yields the language equation $A \diamond X \subseteq C \cup \overline{(IO)^*}$ whose solution S becomes empty under prefix-closure, because S does not contain ϵ, even though it contains the string uv. Thus there is no solution.*

By Proposition 4.3, it is easy to derive an FSM M_{SFSM} associated to S^{FSM}. This allows us to talk about FSMs that are solutions of FSM equations, meaning any reduction of the FSM M_{SFSM}, as guaranteed by Prop. 4.4.

Example 4.8 *Consider the equation $M_A \diamond M_X \preceq M_C$, with the language of M_A, i.e., $A = L_\tau^{\cup}(M_A)$, and the language of M_C, i.e., $C = L_\tau^{\cup}(M_C)$, represented by the automata shown, respectively, in Fig. 8(a) and 8(b). The automata generating the largest language solution, $\overline{(A \cap (\overline{C} \cap (IO)^*)_{\Uparrow U \cup V})_{\Downarrow U \cup V}} \cap (UV)^*$ and its largest prefix closure are portrayed, respectively, in Fig. 8(c) and 8(d).*

Figs. 9 and 10 show the intermediate steps of the computation. Notice that in Fig. 8(d) there are two don't care states, dc1 *non-accepting and* dc2 *accepting, obtained by "splitting" the accepting* dc *state in Fig. 10(b), due to the intersection with the automaton of $(UV)^*$.*

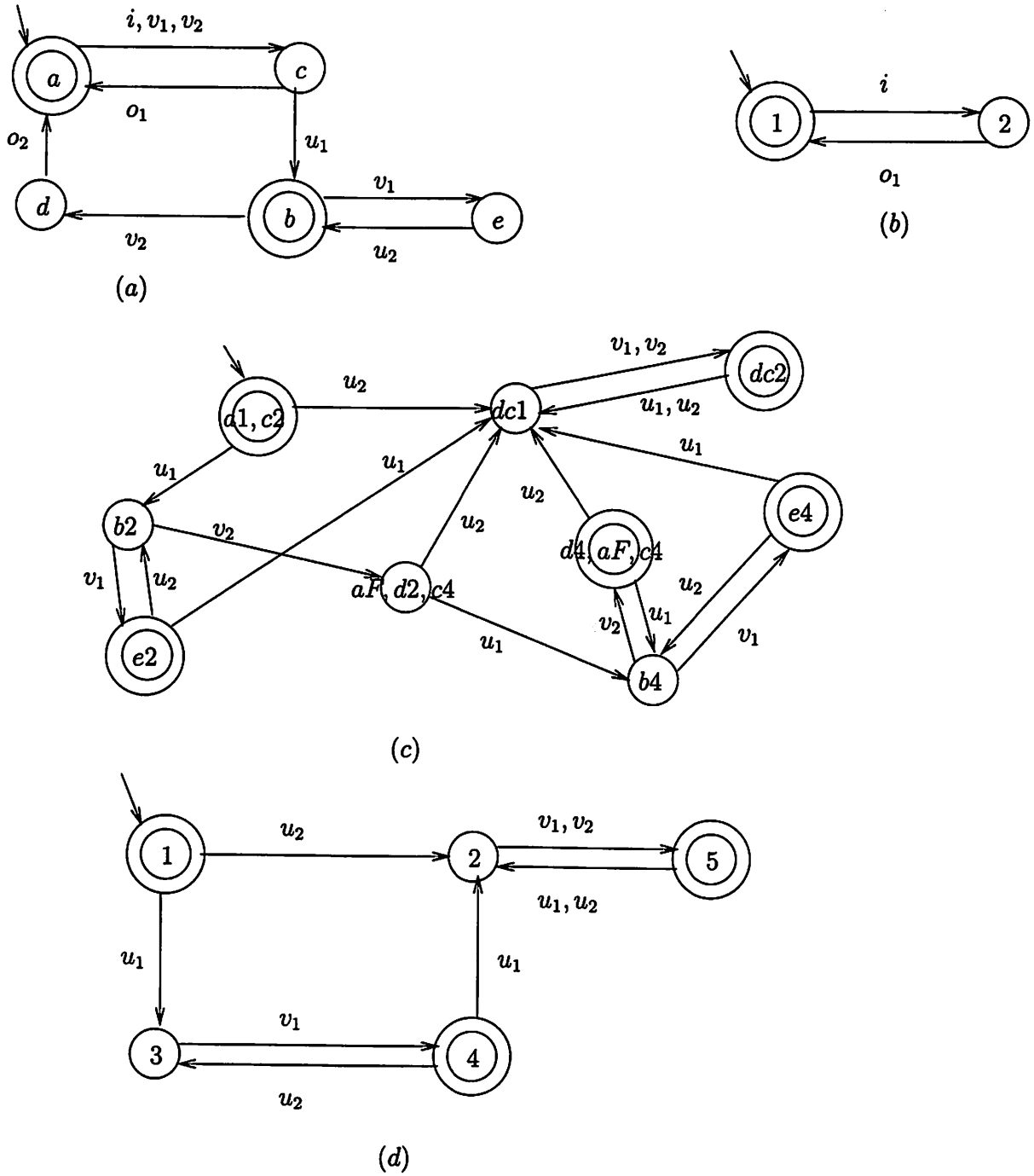

Figure 8: Illustration of Example 4.8. (a) FA of $A = L_r^{\cup}(M_A)$; (b) FA of $C = L_r^{\cup}(M_C)$; (c) FA of $\overline{(A \cap (\overline{C} \cap (IO)^*)_{\Uparrow U \cup V})_{\Downarrow U \cup V}} \cap (UV)^*$; (d) FA of prefix-closure of $\overline{(A \cap (\overline{C} \cap (IO)^*)_{\Uparrow U \cup V})_{\Downarrow U \cup V}} \cap (UV)^*$.

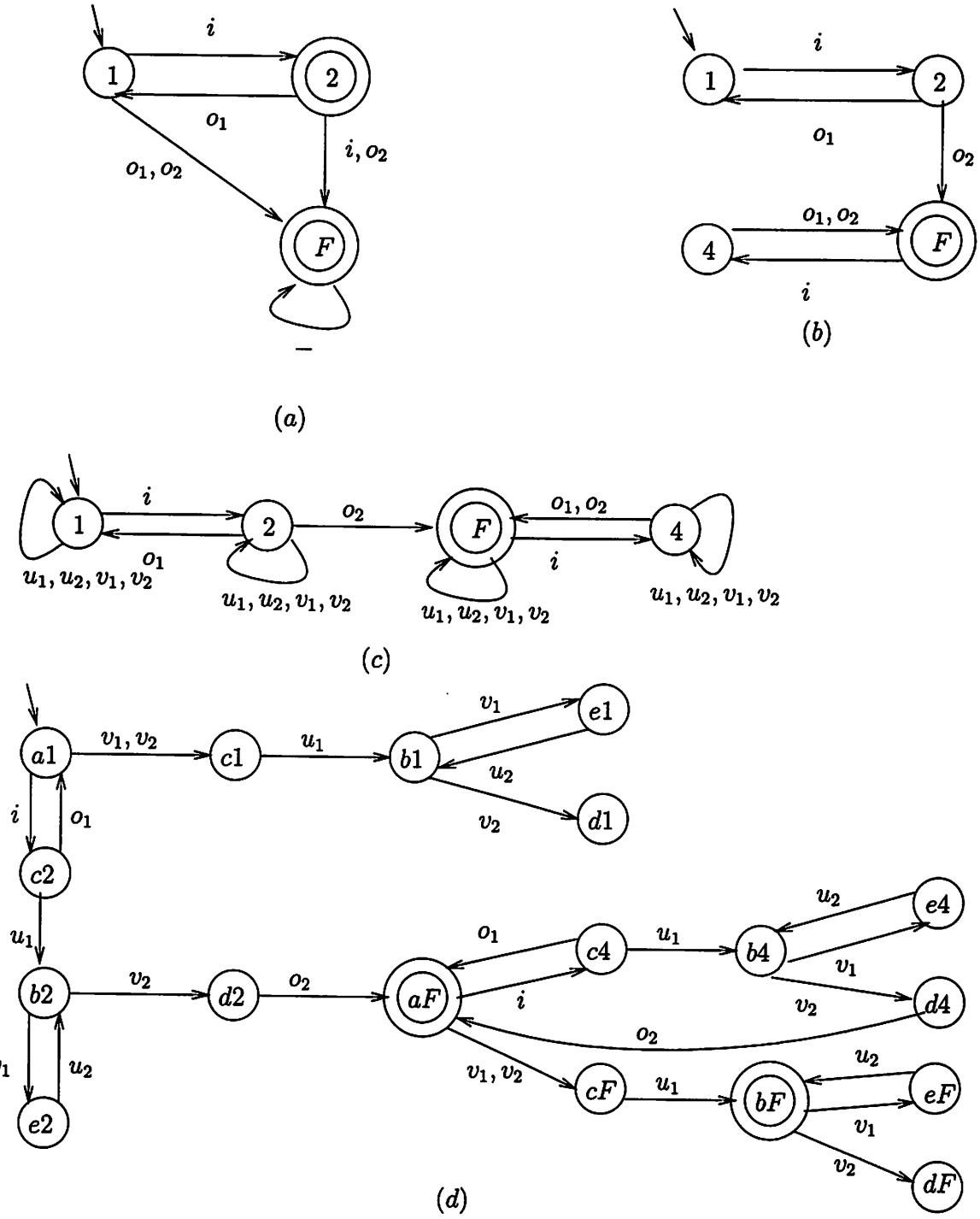

Figure 9: Illustration of Example 4.8. (a) FA of $\overline{C}$; (b) FA of $\overline{C} \cap (IO)^*$; (c) FA of $(\overline{C} \cap (IO)^*)_{\Uparrow U \cup V}$; (d) FA of $A \cap (\overline{C} \cap (IO)^*)_{\Uparrow U \cup V}$.

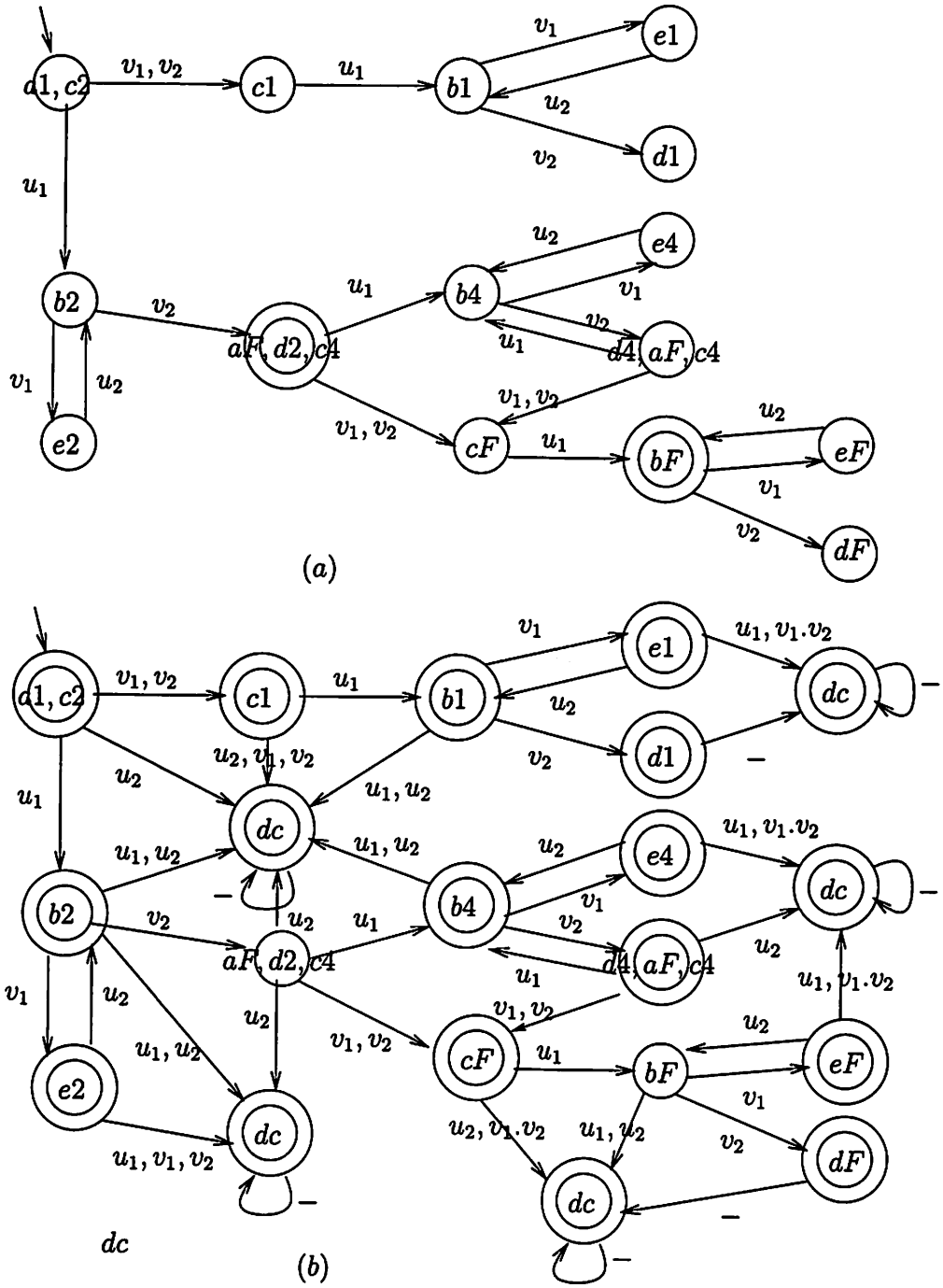

Figure 10: Illustration of Example 4.8. (a) FA of $(A \cap (\overline{C} \cap (IO)^*)_{\Uparrow U \cup V})_{\Downarrow U \cup V}$; (b) FA of $\overline{(A \cap (\overline{C} \cap (IO)^*)_{\Uparrow U \cup V})_{\Downarrow U \cup V}}$.

For logic synthesis applications, we assume that M_A and M_C are complete FSMs and we require that the solution is a complete FSM too. This is obtained by applying Procedure 4.4 to S^{FSM}, yielding $Prog(S^{FSM})$, the largest $(I_2 \cup U)(V \cup O_2)$-progressive FSM language $\subseteq ((I_2 \cup U)(V \cup O_2))^*$.

Proposition 4.7 *FSM M_B is a solution of the equation $M_A \diamond M_X \preceq M_C$, where M_A and M_C are FSMs, iff M_B is a reduction of the FSM $M_{S^{FSM}}$ associated to S^{FSM}, where S^{FSM} is obtained by applying Procedure 4.3 to S, where $S = \overline{A \diamond (\overline{C} \cap (IO)^*)}$. If $S^{FSM} = \emptyset$ then no FSM solution exists. The largest complete FSM solution $M_{Prog(S^{FSM})}$ is found, if it exists, by Procedure 4.4. A complete FSM is a solution iff it is a reduction of the largest complete solution $M_{Prog(S^{FSM})}$.*

The worst-case complexity of computing the largest solution of a parallel equation is of $2^{|S_A| \cdot 2^{|S_C|}}$ as for a synchronous equation. The same analysis applies (restriction plays the same role as projection in introducing nondeterminism).

4.6.2 Largest FSM Compositional Solutions

It is interesting to compute the subset of compositionally I^*O-progressive solutions B, i.e., such that $A_{\Uparrow I_2 \cup O_2} \cap B_{\Uparrow I_1 \cup O_1} \cap (IO)^*_{\Uparrow U \cup V}$ is an I^*O-progressive FSM language $\subseteq (I(U \cup V)^*O)^*$. Thus the composition (after restriction to $I \cup O$) is the language of a complete FSM over inputs $I_1 \cup I_2$ and outputs $O_1 \cup O_2$. Since $A_{\Uparrow I_2 \cup O_2} \cap B_{\Uparrow I_1 \cup O_1} \cap (IO)^*_{\Uparrow U \cup V}$ (after restriction to $I \cup O$) is IO-prefix-closed and ence corresponds to a partial FSM, we have to restrict it so that it is also I^*O-progressive, which corresponds to a complete FSM.

If S^{FSM} is compositionally I^*O-progressive, then S^{FSM} is the largest compositionally I^*O-progressive solution of the equation. However, not every non-empty subset of S^{FSM} inherits the feature of being compositionally I^*O-progressive. If S^{FSM} is not compositionally I^*O-progressive, then denote the largest compositionally I^*O-progressive subset of S^{FSM} by $cI^*OProg(S^{FSM})$. Conceptually, the language $cI^*OProg\ (S^{FSM})$ is obtained from S^{FSM} by deleting each string α such that, for some $i \in I$, there is no $(u \cup v)^* \in (U \cup V)^*$ and no $o \in O$ for which $\alpha\, i(u \cup v)^* o \in A_{\Uparrow I_2 \cup O_2} \cap S^{FSM}_{\Uparrow I_1 \cup O_1} \cap (IO)^*_{\Uparrow U \cup V}$ holds. We expect that a procedure to compute the largest compositionally I^*O-progressive prefix-closed solution, $cProg(S^{FSM})$, can be designed following the pattern of Proc. 4.6, but as yet have not worked out the details. A procedure to compute the largest compositionally progressive solution of a parallel equation over regular languages for the rectification topology was provided in [20].

To characterize subsets of solutions well-behaved with respect to deadlocks and livelocks (endless cycles of internal actions), we introduce a few more language definitions.

Definition 4.14 *A solution B of Eq. 10 is A-***compositionally prefix I^*O-progressive** *if*

$$Init(A)_{\Uparrow I_2 \cup O_2} \cap Init(B)_{\Uparrow I_1 \cup O_1} \cap Init((IO)^*)_{\Uparrow U \cup V}$$

is I^*O-progressive.

A compositionally prefix I^*O-progressive solution yields a composition that allows $(u \cup v)^*$ cycles without exit, yet every sequence in I^*O followed by an input in I must be followed by a $(u \cup v)^*$ cycle that can be exited (by an output).

Definition 4.15 *A solution B of Eq. 10 is A-***compositionally prefix $(U \cup V)$-deadlock-free** *if*

$$Init(A)_{\Uparrow I_2 \cup O_2} \cap Init(B)_{\Uparrow I_1 \cup O_1} \cap Init((IO)^*)_{\Uparrow U \cup V}$$

is $(U \cup V)$-deadlock-free.

A compositionally prefix $(U \cup V)$-deadlock-free solution yields a composition that has no $(u \cup v)^*$ cycles without exit.

Definition 4.16 *A solution B of Eq. 10 is A-***compositionally prefix** $(U \cup V)$***-convergent** *if*

$$Init(A)_{\Uparrow I_2 \cup O_2} \cap Init(B)_{\Uparrow I_1 \cup O_1} \cap Init((IO)^*)_{\Uparrow U \cup V}$$

is $(U \cup V)$-convergent.

A compositionally prefix $(U \cup V)$-convergent solution yields a composition that has no $(u \cup v)^*$ cycles, i.e., it is livelock-free. A compositionally prefix $(U \cup V)$-deadlock-free solution does not need to be compositionally prefix $(U \cup V)$-convergent.

Example 4.9 *Consider the equation $M_A \diamond M_X \preceq M_C$, where FSMs M_A and M_C and the largest solution M_B are shown in Fig. 11(a)-(b)-(c). Fig. 11(d)-(e) shows the related automata A and $B_{\Uparrow I_1 \cup O_1} = B_{\Uparrow \{i\} \cup \{o\}}$, whereas Fig. 11(f)-(g) portrays the automata representing the languages $A_{\Uparrow I_2 \cup O_2} \cap B_{\Uparrow I_1 \cup O_1} \cap (IO)^*_{\Uparrow U \cup V} = A \cap B_{\Uparrow \{i\} \cup \{o\}} \cap (IO)^*_{\Uparrow \{u\} \cup \{v\}}$ and $Init(A)_{\Uparrow I_2 \cup O_2} \cap Init(B)_{\Uparrow I_1 \cup O_1} \cap Init((IO)^*)_{\Uparrow U \cup V} = Init(A) \cap Init(B)_{\Uparrow \{i\} \cup \{o\}} \cap Init((IO)^*)_{\Uparrow \{u\} \cup \{v\}}$.*

If FSM M_A answers by u to the external input i then FSMs M_A and M_B fall into an infinite dialogue, so we would like to classify their composition as neither $(U \cup V)$-convergent nor $(U \cup V)$-deadlock-free. However the language $A \cap B_{\Uparrow \{i\} \cup \{o\}} \cap (IO)^*_{\Uparrow \{u\} \cup \{v\}} = \{(io)^*\}$ is both $(U \cup V)$-convergent and $(U \cup V)$-deadlock-free. To overcome this modeling problem, we introduce the operator Init (guarantees prefix-closure) and rewrite the previous language composition as $Init(A) \cap Init(B)_{\Uparrow \{i\} \cup \{o\}} \cap Init((IO)^*)_{\Uparrow \{u\} \cup \{v\}}$. The latter language is neither $(U \cup V)$-convergent (since $i_{\Uparrow \{u\} \cup \{v\}}$ includes $iu(v+u)^*$ that is a subset of the language) nor $(U \cup V)$-deadlock-free (αiu cannot be extended to a string ending by o, against the definition of $(U \cup V)$-deadlock-free).

Finally 11(h) shows a language that is $(U \cup V)$-deadlock-free, but not $(U \cup V)$-convergent.

Theorem 4.12 *Let B be an $(I_2 \cup U)(V \cup O_2)$-progressive solution of $A \diamond X \subseteq C \cup \overline{(IO)^*}$ and let A be $(I_1 \cup V)(U \cup O_1)$-progressive. If B is compositionally prefix $(U \cup V)$-convergent, then B is compositionally prefix $(U \cup V)$-deadlock-free.*

Proof. Since the components A and B are progressive, their composition $Init(A)_{\Uparrow I_2 \cup O_2} \cap Init(B)_{\Uparrow I_1 \cup O_1} \cap Init((IO)^*)_{\Uparrow U \cup V}$ is deadlock-free, i.e., it never stops because a component does not have a transition under a given input. If the composition is also $(U \cup V)$-convergent, there can be no livelocks, i.e., there are no cycles labeled with actions from the set $U \cup V$. Therefore an external input, after a finite path labelled with internal actions, must be followed by an external output. □

The computation of the largest subset of compositionally prefix $(U \cup V)$-deadlock-free solutions and of the largest subset of compositionally prefix $(U \cup V)$-convergent solutions requires further investigation. The former problem appears similar to the one of finding the largest subset of compositionally I^*O-progressive solutions. About the latter problem, when S^{FSM} is not compositionally prefix $(U \cup V)$-convergent, then the largest compositionally prefix $(U \cup V)$-convergent solution does not exist and each finite IO-prefix-closed subset of S^{FSM} is a compositionally prefix $(U \cup V)$-convergent solution. It is an open question whether there is the largest complete prefix $(U \cup V)$-convergent solution.

4.6.3 FSM Equations under Bounded Parallel Composition

Here we discuss the solutions whose composition with the context produces an external output after at most l internal actions. One could build an analogy with Moore solutions of synchronous equations. We provide in the sequel the key steps to solve FSM equations under bounded parallel composition.

Definition 4.17 *The l-***bounded parallel composition of FSMs** *M_B, over input alphabet $I_2 \cup U$ and output alphabet $O_2 \cup V$, with M_A, over input alphabet $I_1 \cup V$ and output alphabet $O_1 \cup U$, yields the FSM $M_A \diamond_l M_B$ with language*

$$\begin{aligned} L(M_A \diamond_l M_B) &= L(M_A) \diamond_l L(M_B) \cap (IO)^* \\ &= [L(M_A)_{\Uparrow I_2 \cup O_2} \cap L(M_B)_{\Uparrow I_1 \cup O_1} \cap (I \cup O)^*_{\Uparrow (U \cup V, l)}]_{\Downarrow I \cup O} \cap (IO)^*. \end{aligned}$$

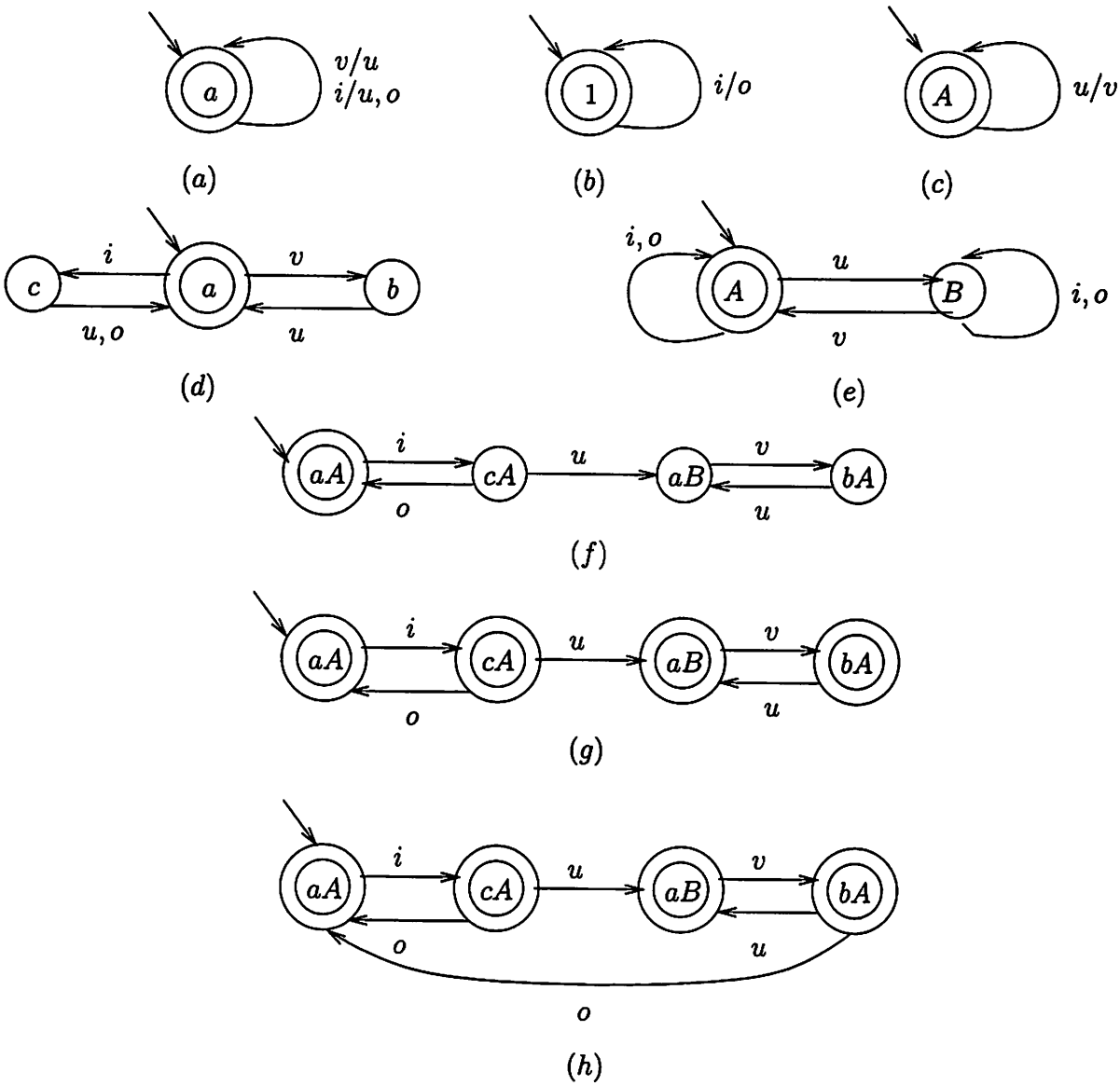

Figure 11: Illustration of Example 4.9. (a) FSM M_A; (b) FSM M_C; (c) FSM M_B; (d) FA of $A = L_r^{\cup}(M_A)$; (e) FA of $B_{\Uparrow\{i\}\cup\{o\}}$, where $B = L_r^{\cup}(M_B)$; (f) FA of $A \cap B_{\Uparrow\{i\}\cup\{o\}} \cap (IO)^{\star}_{\Uparrow\{u\}\cup\{v\}}$; (g) FA of $Init(A) \cap Init(B)_{\Uparrow\{i\}\cup\{o\}} \cap Init((IO)^{\star})_{\Uparrow\{u\}\cup\{v\}}$; (h) FA of prefix $(U \cup V)$-deadlock-free, but not prefix $(U \cup V)$-convergent language.

When $l = \infty$, it reduces to the definition of parallel composition of FSMs.

Proposition 4.8 *FSM M_B is a solution of the equation $M_A \diamond_l M_X \preceq M_C$, where M_A and M_C are FSMs, iff M_B is a reduction of the FSM $M_{S^{FSM}}$ associated to S^{FSM}, where S^{FSM} is obtained by applying Procedure 4.3 to S, where $S = \overline{(A_{\Uparrow I_2 \cup O_2} \cap (\overline{C} \cap (IO)^*)_{\Uparrow (U \cup V, l)})_{\Downarrow I_2 \cup U \cup V \cup O_2}}$. If $S^{FSM} = \emptyset$ then no FSM is a solution. S^{FSM} is the largest compositionally $(U \cup V)$-convergent solution of $M_A \diamond_l M_X \preceq M_C$. The largest complete FSM solution $M_{Prog(S^{FSM})}$ is found, if it exists, by Procedure 4.4.*

Theorem 4.13 *A solution M_B of $M_A \diamond_l M_X \preceq M_C$ is also a compositionally $(U \cup V)$-convergent solution of $M_A \diamond M_X \preceq M_C$.*

If M_A and M_B are also complete FSMs, then M_B is a compositionally prefix I^*O-progressive and compositionally I^*O-progressive solution of $M_A \diamond M_X \preceq M_C$.

Proof. By construction, a solution M_B of $M_A \diamond_l M_X \preceq M_C$ is compositionally $(U \cup V)$-convergent. A solution M_B of $M_A \diamond_l M_X \preceq M_C$ is also a solution of $M_A \diamond M_X \preceq M_C$, because when $l = \infty$ the operator $\diamond_l$ becomes the operator $\diamond$.

By Theorem 4.12, the fact that M_B is compositionally $(U \cup V)$-convergent, together with the completeness of M_A and M_B, imply that M_B is compositionally prefix I^*O-progressive and therefore compositionally I^*O-progressive. □

However, in general $M_A \diamond M_X \preceq M_C$ may be solvable despite the fact that $M_A \diamond_l M_X \preceq M_C$ has no solution. For instance, this may happen when $M_A \diamond M_X \preceq M_C$ has no compositionally I^*O-progressive solution. If the equation $M_A \diamond_l M_X \preceq M_C$ has no complete solution, it is open whether there is a compositionally I^*O-progressive solution of $M_A \diamond M_X \preceq M_C$.

5 An Example: Solving the Protocol Mismatch Problem

A communication system has a sending part and a receiving part that exchange data through a specific protocol. A mismatch occurs when two systems with different protocols try to communicate. The mismatch problem is solved by designing a converter that translates between the receiver and the sender, while respecting the overall service specification of the behaviour of the composed communication system relative to the environment. We formulate the problem as a parallel language equation: given the service specification C of a communication system, a component sender and a component receiver, find a converter X whose composition with the sender and receiver A meets the system specification after hiding the internal signals: $A \diamond X \subseteq C$.

As an example we consider the problem of designing a protocol converter to interface: an *alternating-bit* (AB) sender and a *non-sequenced* (NS) receiver. This problem is adapted from [20] and [12]. A communication system based on an alternating bit protocol is composed of two processes, a sender and a receiver, which communicate over a half duplex channel that can transfer data in either directions, but not simultaneously. Each process uses a control bit called the alternating bit, whose value is updated by each message sent over the channel in either direction. The acknowledgement is also based on the alternating bit: each message received by either process in the system corresponds to an acknowledgement message that depends on the bit value. If the acknowledgement received by a process does not correspond to the message sent originally, the message is resent until the correct acknowledgement is received. On the other hand, a communication system is non-sequenced when no distinction is made among the consecutive messages received or their corresponding acknowledgements. This means that neither messages nor their acknowledgements are distinguished by any flags such as with the alternating bit.

Fig. 12 shows the block diagram of the composed system. Each component is represented by a rectangle with incoming and outgoing labeled arrows to indicate the inputs and outputs, respectively. The sender consists of an AB protocol sender (*PS*) and of an AB protocol channel (*PC*). Meanwhile, the receiving part includes an NS protocol receiver (*PR*). The converter X must interface the two mismatched protocols and

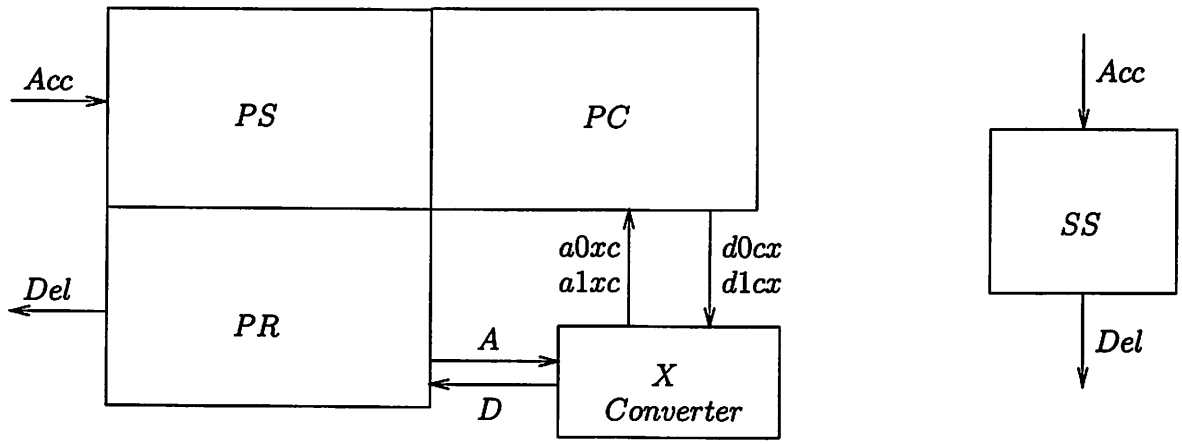

Figure 12: Communication system described in Sec. 5.

guarantee that its composition with PS, PC and PR refines the service specification (SS) of the composed system. The events *Acc* (*Accept*) and *Del* (*Deliver*) represent the interface of the communication system with the environment (the user). The converter X translates the messages delivered by the sender PS (using the alternating bit protocol) into a format that the receiver PR understands (using the non-sequenced protocol). For example, acknowledgement messages A delivered to the converter by the receiver are transformed into acknowledgements of the alternating bit protocol (*a0xc* to acknowledge a 0 bit and *a1xc* to acknowledge a 1 bit) and passed to the sender by the channel (*a0cs* to acknowledge a 0 bit and *a1cs* to acknowledge a 1 bit); data messages are passed from the sender to the channel (*d0sc* for a message controlled by a 0 bit and *d1sc* for a message controlled by a 1 bit) and then from the channel to the converter (*d0cx* for a message controlled by a 0 bit and *d1cx* for a message controlled by a 1 bit) to be transformed by the converter into a data message D for the receiver.

We model the components as I/O automata [23], which recognize prefix-closed regular languages, and we solve their language equations. Fig. 13 shows the automata of the components of the communication system. Missing transitions go to a trap (non-accepting) state, that loops to itself under any event.

Fig. 14 shows the largest prefix-closed solution $S = \overline{PS \diamond PC \diamond PR \diamond \overline{SS}}$ of the converter problem. Notice that all missing transitions go to an *accepting* trap state *dc*, that loops to itself under any event; e.g., the initial state has a transition to state *dc* under events $A, a0xc, a1xc, d1cx$. These transitions are not indicated in the state transition graph of the automaton of the solution language to avoid cluttering the picture. State *dc* can be termed the *don't care* state, because it is introduced during the determinization step to complete the automaton $PS \diamond PC \diamond PR \diamond \overline{SS}$, before the final complementation. It is reached by transitions that cannot occur due to impossible combinations of events in the composition of $PS \diamond PC \diamond PR$ and S, and so it does not matter how S behaves, once it is in state *dc* (thus the qualification *don't care* state). This makes the largest solution S non-deterministic. The solution presented in [20] and [12] does not feature this trap accepting state and so it is not complete (in [20] and [12] all missing transitions of the solution are supposed to end up in a *non-accepting* trap state, a *fail* state); without the above *dc* state, one gets only a subset of all solutions.

Fig. 15 shows another view of the largest prefix-closed solution $S = \overline{PS \diamond PC \diamond PR \diamond \overline{SS}}$ of the converter problem, with the *dc* state included and the *fail* state excluded.

Fig. 16 shows the largest prefix-closed 2-bounded solution of the converter problem.

Fig. 17 shows the composition $PS \diamond PC \diamond PR \cap S_{\uparrow\uparrow\{Acc,Del\}}$ of the communication system $PS \diamond PC \diamond PR$ and of the largest converter S. The largest prefix-closed solution S is compositionally (I^*O)-progressive and compositionally prefix $(U \cup V)$-deadlock-free, but not compositionally prefix $(U \cup V)$-convergent.

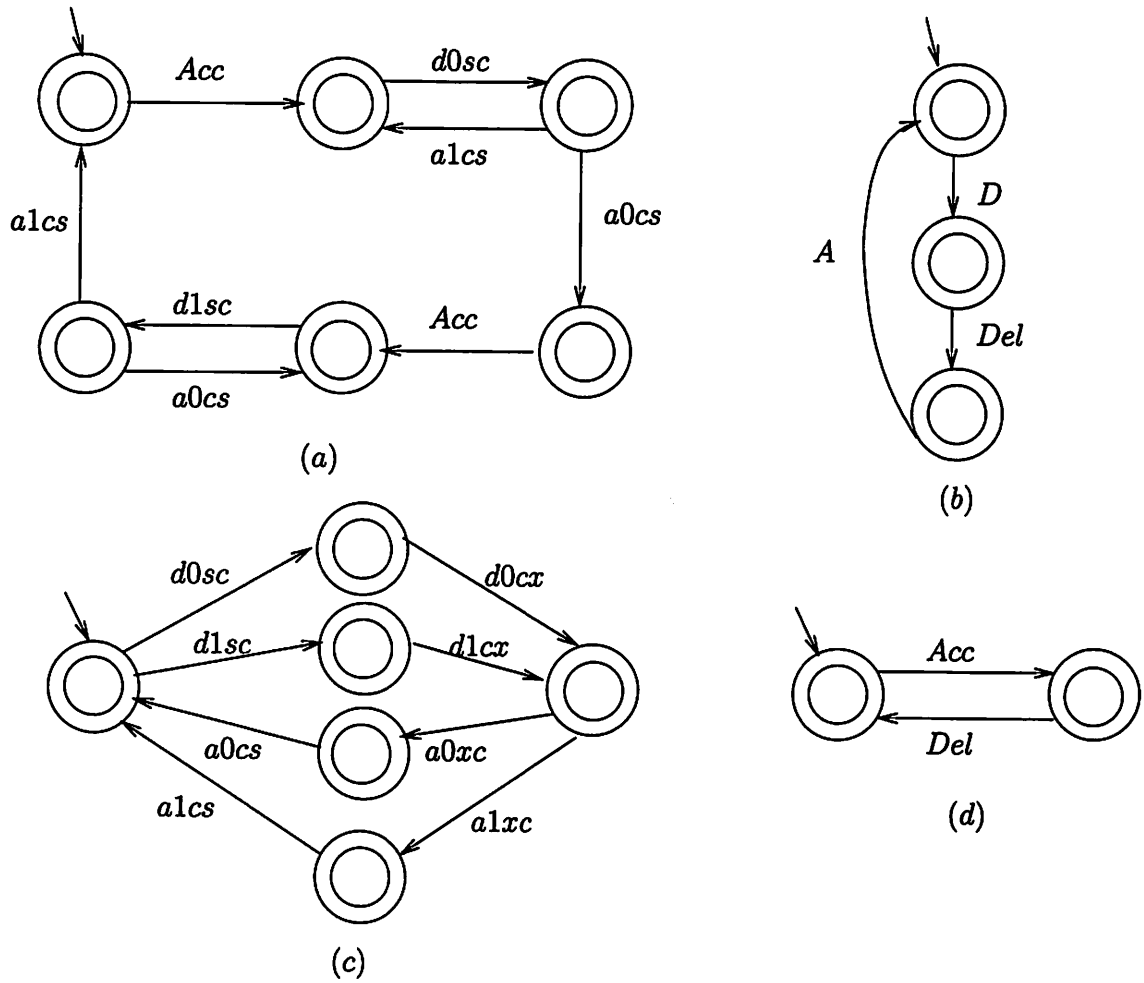

Figure 13: Automata of communication system described in Sec. 5 (a) Automaton of PS; (b) Automaton of PR; (c) Automaton of PC; (d) Automaton of SS.

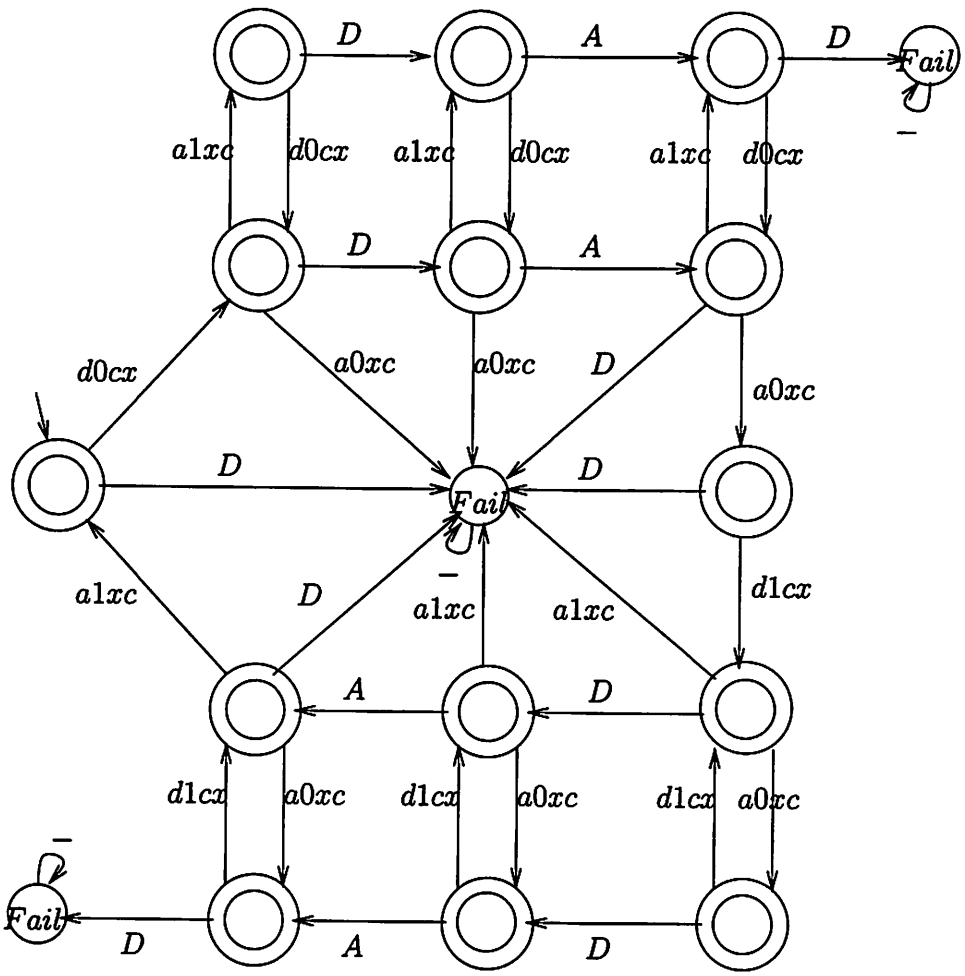

Figure 14: Largest prefix-closed solution $S = \overline{PS \diamond PC \diamond PR \diamond \overline{SS}}$ of the converter problem of Sec. 5.

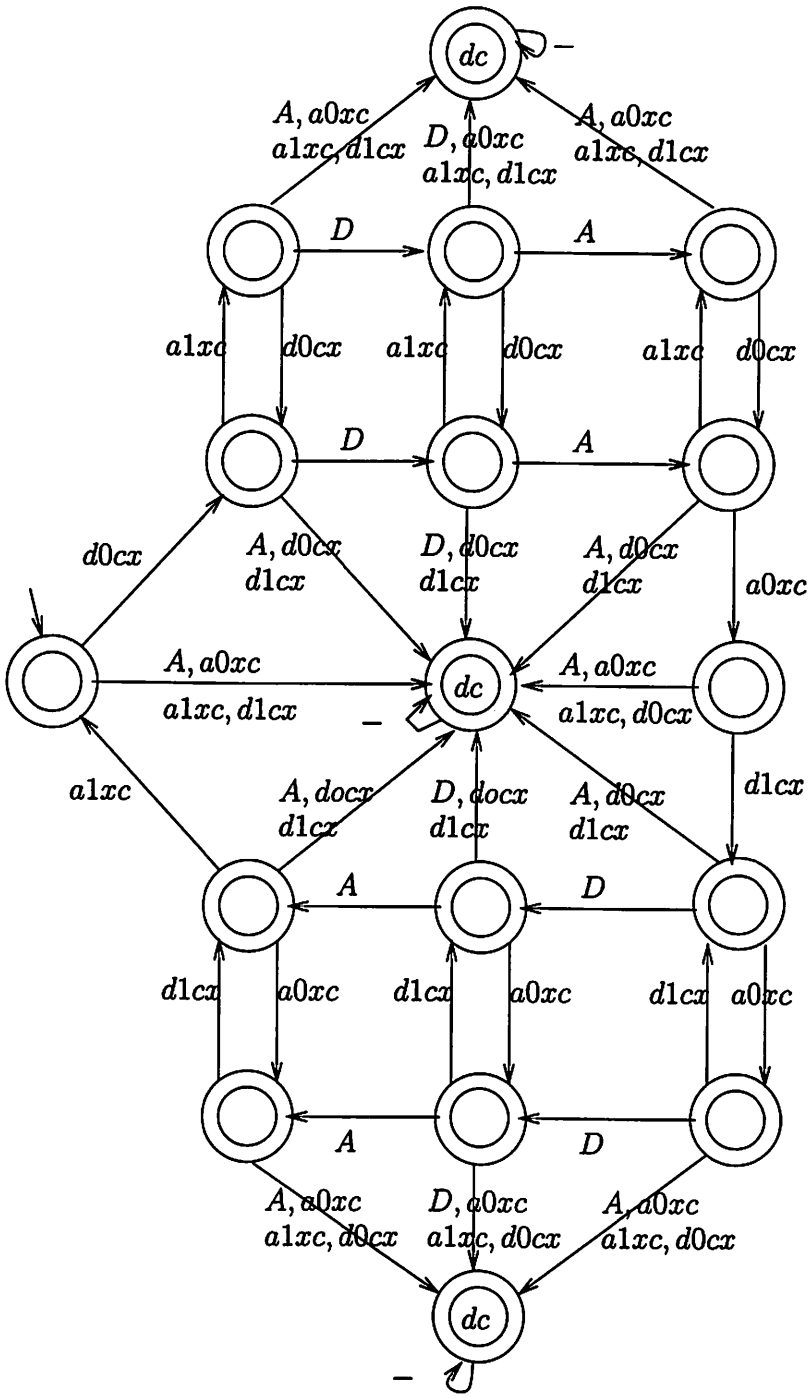

Figure 15: Largest prefix-closed solution S of the converter problem of Sec. 5. It shows explicitly the transitions to the *dc* state.

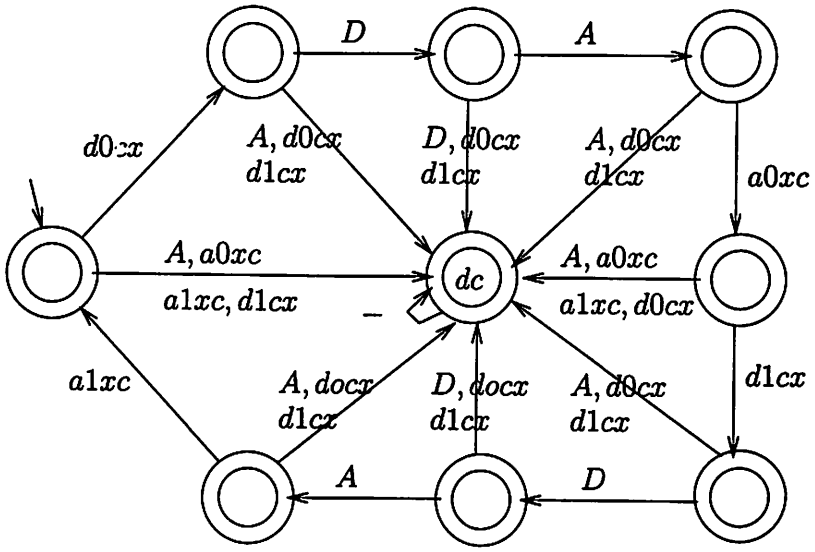

Figure 16: Largest prefix-closed 2-bounded solution of the converter problem of Sec. 5.

6 Comparison with Previous Approaches

6.1 Equations under Synchronous Composition

Sequential synthesis offers a collection of problems that can be modeled by FSM equations under synchronous composition. Some have been attacked in the past with various techniques in different logic synthesis applications.

Hierarchical Optimization and Don't Care Sequences

The goal of hierarchical optimization is to optimize the FSMs of a network capturing the global network information by means of don't care conditions.

This paradigm generalizes what is done already in multi-level combinational synthesis [9, 7], where a lot of effort has been invested in capturing the don't cares conditions and devising efficient algorithms to compute them or their subsets. In particular *input controllability don't cares* and *output observability don't cares* [8] have been defined for multi-level combinational networks (see [26] for an introduction to the topic).

When the theory is extended to sequential circuits, don't care sets become sequences of inputs instead of single inputs, since sequential circuits transform input sequences into output sequences [9].

Input Don't Care Sequences

Consider a cascade interconnection of two FSMs M_1 and M_2, where the driving FSM M_1 feeds the input patterns to the driven FSM M_2. Then input controllability don't cares are the sequences of outputs not produced by M_1: they restrict the controllability of the driven FSM M_2 and are used to modify M_2 obtaining an FSM $\hat{M}_2$ such that the cascade interconnection does not change, i.e., $M_1 \rightarrow \hat{M}_2 = M_1 \rightarrow M_2$. Kim and Newborn [17] were the first to give a procedure to compute all input controllability don't care sequences for a series topology. Later on, H.-Y. Wang [36] showed that input don't care sequences for a component in a network of FSMs with an arbitrary topology can be exploited in the same way as in a series topology and that computing input don't care sequences for an arbitrary topology can be reduced to computing them for a series topology. The theory of input don't care sequences was developed independently in Russia by

[8] **The input controllability don't care set,** CDC_{in}, includes all input patterns that are never produced by the environment at the network's inputs. **The output observability don't care sets,** ODC_{out}, denote all input patterns that represent situations when an output is not observed by the environment.

[9] **The input controllability sequential don't care set,** CDC_{in}^{seq}, includes all input sequences that are never produced by the environment at the network's inputs. **The output observability sequential don't care sets,** ODC_{out}^{seq}, denote all input sequences that represent situations when an output is not observed by the environment at the current time or in the future.

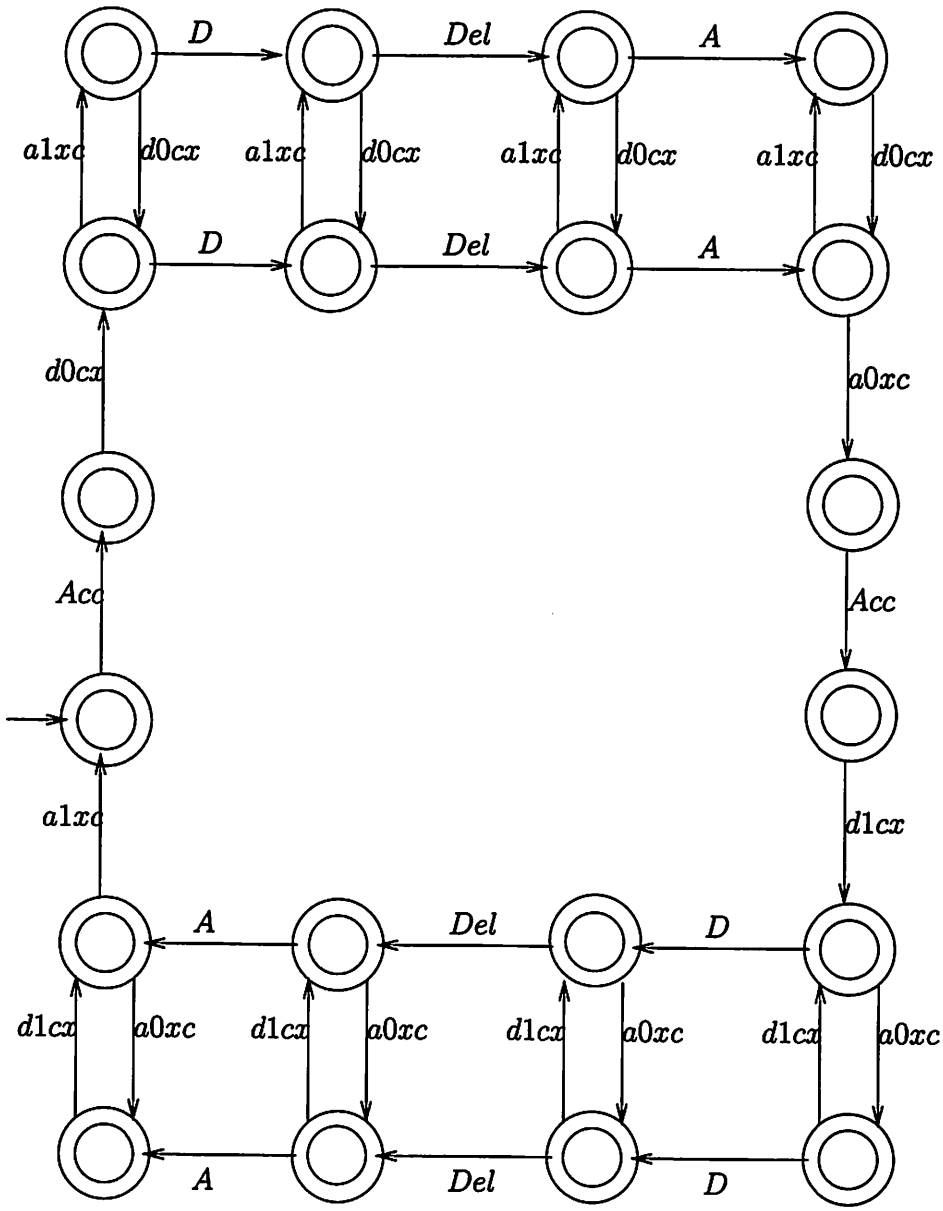

Figure 17: Composition $PS \diamond PC \diamond PR \cap S_{\Uparrow\{Acc,Del\}}$ of communication system $PS \diamond PC \diamond PR$ and largest converter S of the converter problem of Sec. 5.

Yevtushenko [41].

Output Don't Care Sequences

Output observability don't cares are the sets of sequences of inputs of M_2 that cannot be distinguished by the outputs of M_2, i.e., the sequences of a set cannot be distinguished from each other by looking to the outputs of M_2: they restrict the observability of the driving FSM M_1 and are used to modify M_1 obtaining an FSM $\hat{M}_1$ such that the cascade interconnection does not change, i.e., $\hat{M}_1 \rightarrow M_2 = M_1 \rightarrow M_2$. An interesting procedure to compute a subset of the sequential output don't cares is due to H.-Y. Wang [37]. For a survey on the topic refer to [13].

The complete flexibility for the head FSM of a series composition was derived by Yevtushenko and Petrenko [41, 31, 30] by means of a NDFSM whose states are the cartesian product of the components' states, and whose transition relation is unspecified for the inputs such that no internal signal produces the reference output, otherwise it includes all transitions with allowed internal signals.

The first result [41] solved the special case of Moore FSMs where the tail component produces different outputs for different states. Consider a series composition $M_A \rightarrow M_B$ of two Moore FSMs $M_A = (S_A, I, U, \delta_A, \lambda_A, r_A)$ and $M_B = (S_B, U, O, \delta_B, \lambda_B, r_B)$, such that $\forall s_1, s_2 \in S_B\ s_1 \neq s_2$ implies $\lambda_B(s_1) \neq \lambda_B(s_2)$. The FSM representing all behaviours that can be realized at the head component is given by the NDFSM $M_D = (S_A \times S_B, I, U, \delta_D, \lambda_D, (r_A, r_B))$, where $\delta_D((s_A, s_B), i) = (\delta_A(s_A, i), \delta_B(s_B, \lambda_A(s_A)))$ and $\lambda_D(s_A, s_B) = \{u \mid \delta_B(s_B, y) = \delta_B(s_B, \lambda_A(s_A))\}$, i.e., the output of M_D at state (s_A, s_B) is the set of $u \in U$ that drive M_2 from s_B into the same state to which $\lambda_A(s_A)$ does.

Theorem 6.1 *$M_C \rightarrow M_B = M_A \rightarrow M_B$ iff M_C is a reduction of M_D.*

The method was then extended to arbitrary tail machines through two more contributions [31, 30]. The first one [31] proposes an algorithm for output don't care sequences dual to the one by Kim and Newborn for input don't care sequences.

Consider a series composition $M_A \rightarrow M_B$ of two FSMs $M_A = (S_A, I, U, \delta_A, \lambda_A, r_A)$ and $M_B = (S_B, U, O, \delta_B, \lambda_B, r_B)$. The FSM representing all classes of input sequences equivalent with respect to M_B (M_B produces the same output sequence under these input sequences, which are the don't care output sequences of A) is given by the NDFSM $M_D = (S_B \times S_B, U, U, T, (r_B, r_B))$, where the transition $((\hat{s}_B, \tilde{s}_B), u_1, u_2, (\hat{s}'_B, \tilde{s}'_B)) \in T$ iff $\lambda_B(\hat{s}_B, u_1) = \lambda_B(\tilde{s}_B, u_2)$, $\hat{s}'_B = \delta_B(\hat{s}_B, u_1)$ and $\tilde{s}'_B = \delta_B(\tilde{s}_B, u_2)$. In other words, the output sequences produced by FSM M_D under a given input sequence α are those under which M_A produces the same output sequence that it generates under α.

Theorem 6.2 *$M_C \rightarrow M_B = M_A \rightarrow M_B$ iff M_C is a reduction of the product of M_A and M_D.*

The latter contribution [30] builds directly the NDFSM capturing all the flexibility. Consider a series composition $M_A \rightarrow M_B$ of two FSMs $M_A = (S_A, I, U, \delta_A, \lambda_A, r_A)$ and $M_B = (S_B, U, O, \delta_B, \lambda_B, r_B)$. The FSM representing all behaviours that can be realized at the head component is given by the NDFSM $M_D = (S_A \times S_B \times S_B, I, U, T, (r_A, r_B, r_B))$, where $((s_A, \hat{s}_B, \tilde{s}_B), i, u, (s'_A, \hat{s}'_B, \tilde{s}'_B)) \in T$ iff the output of $M_A \rightarrow M_B$ at state $(s_A, \hat{s}_B)$ under input i is equal to the output of M_B at state $\tilde{s}_B$ under input u, and $(s'_A, \hat{s}'_B, \tilde{s}'_B)$ are the successor states respectively in $M_A \rightarrow M_B$ and M_B.

Theorem 6.3 *$M_C \rightarrow M_B = M_A \rightarrow M_B$ iff M_C is a reduction of M_D.*

Computation of the Permissible Behaviors with the E-machine

Given the network topology shown in Fig. 1(d), a fixed-point computation was been defined by Watanabe and Brayton in [38] to compute a PNDFSM that contains *all* behaviors M_B (DFSMs) whose composition with the given machine M_A is contained in the specification M_C. The PNDFSM so obtained has been called the *E-machine*, where the prefix E stands for environment. An alternative computation, credited to A. Saldanha, builds an equivalent NDFSM (see [15] for a detailed exposition). The authors have also investigated the issue of logical implementability of the DFSMs contained in the E-machine, i.e., the problem of finding those contained DFSMs M_B such that there exists a pair of circuit implementations of M_B and M_A with no combinational cycles created by connecting them together by the internal signals u and v. These DFSMs are called *permissible.*

FSM Network Synthesis by WS1S

WS1S (Weak Second-Order Logic of 1 Successor) is a logic with the same expressive power as regular languages [5, 35, 18]. The WS1S formalism enables one to write down easily equations that characterize the set of (functionally) permissible behaviors at a node for different topologies, as pointed out by A. Aziz *et al.* in [3]. We present the equations for some FSM networks shown in Figure 1. In all equations M_C represents the specification; in a given equation, we label the unknown FSM by a superscript *. According to the WS1S syntax, the FSMs must be encoded to appear in the equations.

1-Way Cascade (a) - Fig. 1(c) $\phi^{M_A{}^*}(I_1, U) = (\forall O_2)[\phi^{M_B}(U, O_2) \rightarrow \phi^{M_C}(I_1, O_2)]$.

The machine M_A^* is exactly the one produced by the construction due to Kim and Newborn [17].

1-Way Cascade (b) - Fig. 1(c) $\phi^{M_B{}^*}(U, O_2) = (\forall I_1)[\phi^{M_A}(I_1, U) \rightarrow \phi^{M_C}(I_1, O_2)]$.

Supervisory Control - Fig. 1(e) $\phi^{M_B{}^*}(I_2, O_1, V) = \phi^{M_A}(V, O_1) \rightarrow \phi^{M_C}(I_2, O_1)$.

The restriction to Moore solutions was investigated in [2].

2-Way Cascade (a) - Fig. 1(b) $\phi^{M_A{}^*}(I_1, V, U) = (\forall O_2)[\phi^{M_B}(U, V, O_2) \rightarrow \phi^{M_C}(I_1, O_2)]$.

2-Way Cascade (b) - Fig. 1(b) $\phi^{M_B{}^*}(U, V, O_2) = (\forall I_1)[\phi^{M_A}(I_1, V, U) \rightarrow \phi^{M_C}(I_1, O_2)]$.

Rectification (a) - Fig. 1(d) $\phi^{M_B{}^*}(U, V) = (\forall I_1, O_1)[\phi^{M_A}(I_1, V, U, O_1) \rightarrow \phi^{M_C}(I_1, O_1)]$.

Rectification (b) - Fig. 1(d) $\phi^{M_A{}^*}(I_1, V, U, O_1) = \phi^{M_B}(U, V) \rightarrow \phi^{M_C}(I_1, O_1)$.

Going beyond previous *ad hoc* approaches, the fact of embedding logic synthesis problems into WS1S formulas allows one to state them in a common frame, enabling easily proof of correctness and completeness of the proposed solutions. Then the computations are performed upon the related automata applying to them the operations that correspond to the standard logical connectives (however the result may be a regular language that is not an FSM language, an issue addressed by Th. 4.8).

In contrast, the theory of synchronous and parallel equations is built upon the primitive notions of language and language composition, and models naturally a larger spectrum of language equations and their specialized solutions.

Model Matching by Simulation Relations

The model matching problem in control theory seeks to design a controller M_B so that the composition of a plant M_A with M_B matches a given model M_C (see the controller's topology in Fig. 1(e)). A procedure for deriving the largest solution for complete FSMs M_A (DFSM) and M_C (PNDFSM) of the equation $M_A \bullet M_X \preceq^{sim} M_C$ for the discrete model matching problem was proposed in [16, 4], where $\preceq^{sim}$ denotes simulation relation [10] (as opposed to language containment in our approach). Simulation relations in general are stronger than language containment, i.e., a simulation relation implies language containment, but not vice versa.

The use of simulation relation instead of language containment avoids determinization and leads to an algorithm of polynomial complexity bounded by $O(|S_A|.|S_C|.|T_A|.|T_C|)$, where $|S_A|$ ($|S_C|$) is the number of states of M_A (M_C) and $|T_A|$ (T_C) is the size of the transition relation of M_A (M_C). In [4] a solvability condition is given, based on the notion of simulation relation between the automata which generate the possible output sequences produced by an FSM.

[10] $\psi \subseteq S_1 \times S_2$ is a **simulation relation** from an FSM $M_1 = \langle S_1, I, O, T_1, r_1 \rangle$ to an FSM $M_2 = \langle S_2, I, O, T_2, r_2 \rangle$ if

1. $(r_1, r_2) \in \psi$, and
2. $(s_1, s_2) \in \psi \Rightarrow$
   $\{ \forall i \, \forall o \, \forall s_1' \, [(s_1 \xrightarrow{i/o}_{M_1} s_1') \Rightarrow \exists s_2' \, [(s_2 \xrightarrow{i/o}_{M_2} s_2') \wedge (s_1', s_2') \in \psi]] \}$.

If such a ψ exists, we say that M_2 **simulates** M_1, or that M_1 has a simulation into M_2, and denote it by $M_1 \preceq^{sim} M_2$.

6.2 Equations under Parallel Composition

Consider the parallel composition of discrete-event systems A over alphabet $I \cup U$ and B over alphabet $O \cup U$, with pairwise-disjoint alphabets I, O and U. A number of papers in process algebra [25, 32, 33, 28, 20, 12] solves the equation $A \diamond X \approx C$ under various relations $\approx$, where A and C are given processes. In this section we focus on equations defined over process languages, A process language is usually a prefix-closed regular language and is represented as the language (or set of traces) of a labeled transition system (LTS), which is a finite automaton (with ϵ moves) where each state is accepting. Some states can also be marked with a partially ordered set of marks.

In [25] the equation $A \diamond X = C$ was studied over LTS languages and the following results were claimed:

Theorem 6.4 *The maximal solution of $A \diamond X \subseteq C$ over prefix-closed regular languages is given by*

$$S = A \diamond C \setminus A \diamond \overline{C}.$$

Theorem 6.5 *The equation $A \diamond X = C$ is solvable over prefix-closed regular languages iff the language $S = A \diamond C \setminus A \diamond \overline{C}$ is a solution of the equation, i.e., $A \diamond (A \diamond C \setminus A \diamond \overline{C}) = C$.*

Theorem 6.4 is not accurate because the largest solution of $A \diamond X \subseteq C$ is $S = \overline{A \diamond \overline{C}}$, which is obtained by adding to the language $A \diamond C \setminus A \diamond \overline{C}$ each word $\alpha \in (U \cup O)^*$ such that the sets $\alpha_{\Downarrow U}$ and $A_{\Downarrow U}$ are disjoint (these words cannot occur in the composition). Given the equation $A \diamond X \subseteq C$ over prefix-closed regular languages accepted by a finite automaton with nonaccepting states, a procedure leading to the same incomplete solution $S = A \diamond C \setminus A \diamond \overline{C}$ is presented and argued to be maximal in [11]. Notice that the version of Procedure 3.1 for parallel composition handles unrestricted regular languages, so it is more general than the ones in [25] or [11]. Safe and compositionally deadlock-free solutions are considered in [10] and a procedure to derive the largest safe solution is proposed.

A procedure for deriving the largest solution for LTSs A and C of the equation $A \diamond X \preceq^{bisim} C$ was proposed in [32], where $\preceq^{bisim}$ denotes either a strong or weak bisimulation relation (as opposed to language containment in our approach). The authors derive an automaton representing all traces of the parallel composition of A and each sequence over the alphabet $U \cup O$. The final state of each trace of the composition whose $I \cup O$-restriction is not a trace of C is marked as a *bad* state. To compute the largest solution, the automaton is augmented with a designated *don't care* state that accepts all unfeasible sequences, i.e,. those that do not occur in the composition. Finally the $(U \cup O)$-restriction of the automaton is derived and the result is determinized by subset construction, where each subset including a bad state is marked as bad. The largest solution is the subset of all sequences of the $(U \cup O)$-restriction whose runs do not end up in bad states.

In [29], the equation $A \diamond X \subseteq C$ is solved over complete FSMs. It is shown that the solution over LTSs cannot be applied directly to FSMs because in general it is not an FSM language (an FSM language must be also prefix-closed and I-progressive). Moreover, compositionally $(U \cup O)$-convergent solutions are considered. A technique is proposed to find the so-called largest candidate solution by deleting from the largest solution the sequences causing deadlocks in the composition with FSM A. If the obtained solution is compositionally $(U \cup O)$-convergent then it is called the largest solution of the equation; otherwise, the question whether a compositionally $(U \cup O)$-convergent solution exists remains open. In the latter case, as in [28], the authors propose to limit to l the number of internal interactions; if, for a given l, the largest solution of the equation exists then it is compositionally $(U \cup O)$-convergent and also each of its reductions inherits this property. Notice that in Sec. 4.6.3 we extended to regular languages the procedure for determining the solution with a limited number of internal interactions. Another restricted solution to parallel equations over regular languages is considered in [19]. The solution is called the minimally restrictive supervisor, i.e., a supervisor that combined with the context matches the largest specification sublanguage.

The equation $A \diamond X = C$ over regular languages, restricted to the case of the rectification topology to model the protocol conversion problem, was addressed in [20]. The solution was found using the theory of

supervisory control of discrete event systems and looks for a converter sublanguage in the language of A; their technique as it is does not generalize to topologies such that the unknown component depends also on signals that do not appear in the component A. Their solution is of the form $A \diamond C \setminus A \diamond \overline{C}$ (not the largest solution). Moreover, they gave an algorithm to obtain the largest compositionally progressive solution by first splitting the states of the automaton of the unrestricted solution (refining procedure, exponential step due to the restriction operator), and then deleting the states that violate the desired requirement of progressive composition (linear step). The protocol conversion problem was discussed also in [12] with the formalism of input-output automata.

In the context of modeling delay-insensitive processes and their environments, a number of concurrency models use various labelings of states of processes to represent certain properties of states, such as quiescence and error or violation [24, 27]. The existence of state labels requires a stronger semantics than language semantics and leads to a reflection operator further refining the language complementation operator. See [24, 27] for discussions on parallel composition operators for delay-insensitive processes. The largest solution of the equation $P \parallel X \leq R$ is the process $\tilde{}(P \parallel \tilde{} R)$, where $\parallel$ is a composition operator and $\tilde{}$ is a reflection operator, replacing the complementation operator used with language semantics. As with language semantics, such a solution might not be compositionally $(U \cup O)$-convergent; it is also of interest to look for solutions exhibiting a property called *healthness*, capturing correctness properties according to the chosen parallel composition operator [24].

7 Conclusions

The problem of finding an unknown component in a network of components in order to satisfy a global system specification was addressed. Abstract language equations of the type $A \bullet X \subseteq C$ and $A \diamond X \subseteq C$ were investigated, where $\bullet$ and $\diamond$ are operators of language composition. The most general solution was computed and various types of constrained solutions were studied. Then specialized language equations were introduced, such as regular and FSM language equations. The maximal subsets of them closed with respect to various language properties were studied. In particular the largest compositional solutions were studied; the first algorithm to compute the largest compositionally I-progressive solution was given.

This approach unifies in a seamless frame the previous reported techniques and appears capable of modeling problems with various notions of composition and types of language acceptors.

These techniques were applied to a classical synthesis problem of a converter between a given mismatched pair of protocols, using their specifications, as well as those of the channel and of the required service. This problem was also addressed in [20, 12] with supervisory control techniques. We were able to derive the largest solution, and the largest compositionally progressive solution, which were not previously reported in the literature.

Future work includes studying other types of language equations (e.g., as applied to game theory and those modeled by Petri nets) and building a prototype software package to compute the solutions automatically. To assess the practical relevance of the method, we plan to test it in different applicative domains and to determine, for each domain, the level of system "granularity" at which the method is effective. Computing approximations will be the next direction to explore, after that exact techniques have been exploited fully.

8 Acknowledgments

The first author was partly supported by the Russian Ministry of High Education (Grant UR.04.01.018). Both the first and the third author gratefully acknowledge the support of a NATO travel grant (NATO Linkage Grant No. 971217). The second author gratefully acknowledges the support of the MADESSII Project (Italian National Research Council). The fourth author gratefully acknowledges the support of NSERC (Grant OGP0194381).

References

[1] A. Arnold. *Finite transition systems : semantics of communicating systems.* Prentice Hall, 1994.

[2] A. Aziz, F. Balarin, R.K. Brayton, M.D. Di Benedetto, A. Saldanha, and A.L. Sangiovanni-Vincentelli. Supervisory control of finite state machines. In *Proceedings of the International Conference on Computer-Aided Verification*, pages 279–292, July 1995.

[3] A. Aziz, F. Balarin, R.K. Brayton, and A. Sangiovanni-Vincentelli. Sequential synthesis using S1S. *IEEE Transactions on Computer-Aided Design*, 19(10):1149–1162, October 2000.

[4] M. Di Benedetto, A. Sangiovanni-Vincentelli, and T. Villa. Model Matching for Finite State Machines. *IEEE Transactions on Automatic Control*, 46(11):1726–1743, December 2001.

[5] J.R. Büchi. *The collected works of J. Richard Büchi.* Springer-Verlag, 1990.

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

[7] E. Cerny. Controllability and fault observability in modular combinational circuits. *IEEE Transactions on Computers*, vol. C-27(10):896–903, October 1978.

[8] E. Cerny. Verification of I/O trace set inclusion for a class of non-deterministic finite state machines. In *The Proceedings of the International Conference on Computer Design*, pages 526–530, October 1992.

[9] E. Cerny and M. Marin. An approach to unified methodology of combinational switching circuits. *IEEE Transactions on Computers*, vol. C-26(8):745–756, August 1977.

[10] J. Drissi and G. v. Bochmann. Submodule construction for systems of I/O automata. Technical Report #1133, DIRO, Universite' de Montreal, Canada, 1999.

[11] E. Haghverdi and H. Ural. Submodule construction from concurrent system specifications. *Information and Software Technology*, 41(8):499–506, June 1999.

[12] H. Hallal, R. Negulescu, and A. Petrenko. Design of divergence-free protocol converters using supervisory control techniques. In *7th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2000*, volume 2, pages 705–708, December 2000.

[13] S. Hassoun and T. Villa. Optimization of synchronous circuits. In R. Brayton, S. Hassoun, and T. Sasao, editors, *Logic Synthesis and Verification*, pages 225–253. Kluwer, 2001.

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

[15] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. *Synthesis of FSMs: functional optimization.* Kluwer Academic Publishers, 1997.

[16] S. Khatri, A. Narayan, S. Krishnan, K. McMillan, R. Brayton, and A. Sangiovanni-Vincentelli. Engineering change in a non-deterministic FSM setting. In *The Proceedings of the Design Automation Conference*, pages 451–456, June 1996.

[17] J. Kim and M.M. Newborn. The simplification of sequential machines with input restrictions. *IRE Transactions on Electronic Computers*, pages 1440–1443, December 1972.

[18] N. Klarlund. Mona & Fido: The logic-automaton connection in practice. In M. Nielsen and W. Thomas, editors, *Computer science logic : 11th international workshop, CSL '97*, volume 1414 of *LNCS*, pages 311–26. Springer-Verlag, 1998.

[19] R. Kumar, V. Garg, and S.I. Marcus. On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17(3):157–168, September 1991.

[20] 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.

[21] R.P. Kurshan. *Computer-aided verification of coordinating processes.* Princeton University Press, 1994.

[22] R.P. Kurshan, M. Merritt, A. Orda, and S.R. Sachs. Modelling asynchrony with a synchronous model. *Formal Methods in System Design*, vol. 15(no. 3):175–199, November 1999.

[23] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, September 1989.

[24] W.C. Mallon, J.T. Tijmen, and T. Werhoeff. Analysis and applications of the XDI model. In *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 231–242, 1999.

[25] P. Merlin and G. v. Bochmann. On the construction of submodule specifications and communication protocols. *ACM Transactions on Programming Languages and Systems*, 5(1):1–25, January 1983.

[26] G. De Micheli. *Synthesis and optimization of digital circuits.* McGraw-Hill, 1994.

[27] R. Negulescu. Process spaces. In C. Palamidessi, editor, *Proceedings of CONCUR 2000, 11th International Conference on Concurrency Theory*, volume 1877 of *LNCS*, pages 199–213. Springer-Verlag, 2000.

[28] A. Overkamp. Supervisory control using failure semantics and partial specifications. *IEEE Transactions on Automatic Control*, 42(4):498–510, April 1997.

[29] A. Petrenko and N. Yevtushenko. Solving asynchronous equations. In S. Budkowski, A. Cavalli, and E. Najm, editors, *Formal Description Techniques and Protocol Specification, Testing and Verification - FORTE XI/PSTV XVIII '98*, pages 231–247. Kluwer Academic Publishers, November 1998.

[30] A. Petrenko, N. Yevtushenko, and R. Dssouli. Testing strategies for communicating finite state machines. In T. Mizuno, T. Higashino, and N. Shiratori, editors, *IFIP WG 6.1 International Workshop on Protocol Test Systems (7th : 1994 : Tokyo, Japan)*, pages 193–208. Chapman & Hall, 1995.

[31] A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das. Non-deterministic state machines in protocol conformance testing. In O. Rafiq, editor, *IFIP TC6/WG6.1 International Workshop on Protocol Test Systems (6th : 1993 : Pau, France)*, pages 363–378. North-Holland, 1994.

[32] H. Qin and P. Lewis. Factorisation of finite state machines under strong and observational equivalences. *Formal Aspects of Computing*, 3:284–307, Jul.-Sept. 1991.

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

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

[35] J.W. Thatcher and J.B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2:57–82, 1968.

[36] H.-Y. Wang and R.K. Brayton. Input don't care sequences in FSM networks. In *The Proceedings of the International Conference on Computer-Aided Design*, pages 321–328, November 1993.

[37] H.-Y. Wang and R.K. Brayton. Permissible observability relations in FSM networks. In *The Proceedings of the Design Automation Conference*, pages 677–683, June 1994.

[38] Y. Watanabe and R.K. Brayton. The maximum set of permissible behaviors for FSM networks. In *IEEE International Conference on Computer-Aided Design*, pages 316–320, November 1993.

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

[40] N. Yevtushenko, T. Villa, R. Brayton, A. Petrenko, and A. Sangiovanni-Vincentelli. Compositionally progressive solutions of synchronous language equations. In *International Workshop on Logic and Synthesis*, May 2003.

[41] N.V. Yevtushenko and A.Y. Matrosova. Synthesis of checking sequences for automaton networks. *Automatic Control and Computer Sciences*, 25(2):1–4, 1991.