Equisolvability of Series vs. Controller's Topology in Synchronous Language Equations

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

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

[§]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

March 15, 2002

Abstract

Given the problem of finding an unknown component that combined with a known part of a system satisfies a desired specification, we proposed already a solution technique based on defining equations over languages associated to the components of the system. We introduced two composition operators for abstract languages: synchronous composition, $\bullet$, and parallel composition, $\diamond$, and we studied the most general solutions of the language equations $A \bullet X \subseteq C$ and $A \diamond X \subseteq C$ ($\subseteq$ denotes language containment), defining the language operators required to express them. In particular we studied the solutions of the equations defined over finite state machines (FSMs) of the type $M_A \bullet M_X \subseteq M_C$ and $M_A \diamond M_X \subseteq M_C$, where M_A models the context, M_C models the specification and M_X is unknown.

Investigating the solutions for various network topologies of synchronous FSM equations, we have been led to the question on the relation between the solvability of an equation for the series topology and the corresponding equation for the controller's topology. In this note we answer to the question, establishing that, if M_A is a deterministic FSM, then the FSM equation $M_X \bullet M_A \preceq M_C$ is solvable for the series topology with an unknown head component iff it is solvable for the controller's topology. Our proof is constructive, meaning that given a solution M_B for the series topology shows how to build a solution M_D for the controller's topology and viceversa.

1 Introduction

An important step in the design of complex systems is the decomposition of the system into a number of separate components which interact in some well-defined way. In this context, a typical question is how to design a component that combined with a known part of the system, called the context, conforms or satisfies or matches 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. There are three key issues to consider:

- how to model the system and its components and the specification,
- how to model the composition of the components, and
- how to model the notion of a system conforming or satisfying or matching another system.

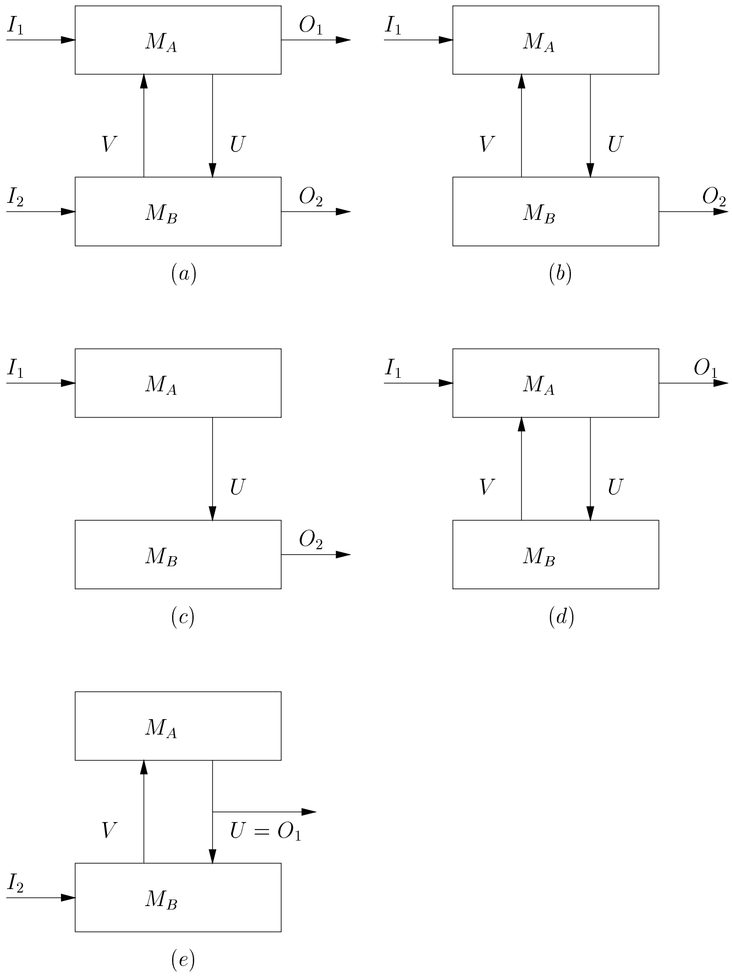

Figure 1: Patterns of composition. (a) general topology; (b) 2-way cascade (or 2-way series) topology; (c) 1-way cascade (or 1-way series) topology; (d) rectification topology; (e) controller's (or supervisory control) topology.

Various contributions investigating partial aspects of the topic of this research have been published in the past. We refer the interested reader to [2], Chap. 6, for a good survey.

In [4] we proposed a general frame based on defining equations over languages associated to the components of a given system. We introduced two composition operators for abstract languages: synchronous composition, $\bullet$, and parallel composition, $\diamond$, and we studied the most general solutions of the language equations $A \bullet X \subseteq C$ and $A \diamond X \subseteq C$ ($\subseteq$ denotes language containment), defining the language operators needed to express them.

Since then we have been developing systemically the theory of equations over languages, considering the languages associated to finite automata. In particular we studied the solutions of the equations defined over finite state machines (FSMs) of the type $M_A \bullet M_X \subseteq M_C$ and $M_A \diamond M_X \subseteq M_C$, where M_A models the context, M_C models the specification and M_X is unknown. A brief summary of synchronous equations over FSMs is provided in the appendices A and B as background of this paper. We refer to [5] for a report on FSM equations of the type $M_A \diamond M_X \subseteq M_C$ (introduced in [3]).

Investigating the solutions for various network topologies of synchronous FSM equations, we have been led to the question on the relation between the solvability of an equation for the series topology and the corresponding equation for the controller's topology. In this note we answer to the question, establishing that, if M_A is a deterministic FSM, then the FSM equation $M_X \bullet M_A \preceq M_C$ is solvable for the series topology with an unknown head component iff it is solvable for the controller's topology. Our proof is constructive, meaning that given a solution M_B for the series topology shows how to build a solution M_D for the controller's topology and viceversa.

2 Equisolvability of Series vs. Controller's Topology

The next theorem argues that a synchronous FSM equation is solvable for the controller's topology iff it is solvable for the series topology. The proof shows how the two sets of solutions relate.

Theorem 2.1 *Given a plant $M_A = (S_A, V, O, T_A, r_A)$ and a reference model $M_C = (S_C, I, O, T_C, r_C)$, if M_A is a DFSM then the FSM equation $M_X \bullet M_A \preceq M_C$ is solvable for the series topology with an unknown head component iff it is solvable for the controller's topology.*

Proof.
Only if Let $M_B = (S, I, V, T_B, r)$ be an FSM solution of the equation for the series topology. Construct the FSM $M_D = (S, I \times O, V, T_D, r)$, where T_D is defined by

$$(s, io, v, s') \in T_D \Leftrightarrow (s, i, v, s') \in T_B.$$

We show that M_D is a solution of the equation for the controller's topology.
Given states $(s, s_A), (s', s'_A) \in S \times S_A$, $i \in I$ and $o \in O$, by definition of $T_{B \bullet A}$,

$$((s, s_A), i, o, (s', s'_A)) \in T_{B \bullet A} \Leftrightarrow \exists v \in V[(s, i, v, s') \in T_B \wedge (s_A, v, o, s'_A) \in T_A].$$

Furthermore, by definition of T_D,

$$\exists v \in V[(s, i, v, s') \in T_B \wedge (s_A, v, o, s'_A) \in T_A] \Leftrightarrow \exists v \in V[(s, io, v, s') \in T_D \wedge (s_A, v, o, s'_A) \in T_A],$$

and, by definition of $T_{D \bullet A}$,

$$\exists v \in V[(s, io, v, s') \in T_D \wedge (s_A, v, o, s'_A) \in T_A] \Leftrightarrow ((s, s_A), i, o, (s', s'_A)) \in T_{D \bullet A}.$$

Summing it up

$$((s, s_A), i, o, (s', s'_A)) \in T_{B \bullet A} \Leftrightarrow ((s, s_A), i, o, (s', s'_A)) \in T_{D \bullet A}.$$

Therefore $M_B \bullet M_A$ and $M_D \bullet M_A$ are equivalent and M_D is a solution of the equation for the controller's topology.

If For notational convenience in this proof we represent the transition relation T of a DFSM by the next state function δ and by the output function λ.

Let $M_D = (S, I \times O, V, \delta_D, \lambda_D, r)$ be a DFSM that is a solution for the controller's topology of the FSM equation $M_X \bullet M_A \preceq M_C$; moreover let the composition $M_D \bullet M_A$ be a DFSM too.

Construct an FSM $B = (S \times S \times S_A, I, V, \delta_B, \lambda_B, (r, r, r_A))$ such that for all $(s, s, s_A) \in S \times S \times S_A$ and $i \in I$ it holds that

$$\begin{aligned}
\delta_B((s, s, s_A), i) &= (\delta_D(s, (i, \lambda_{D \bullet A}((s, s_A), i))), \delta_{D \bullet A}((s, s_A), i))), \\
\lambda_B((s, s, s_A), i) &= \lambda_D(s, (i, \lambda_{D \bullet A}((s, s_A), i))).
\end{aligned}$$

Notice that states of the form (s', s'', s_A), with $s', s'' \in S$ and $s' \neq s''$, need not be considered because they are unreachable. We show that $M_B \bullet M_A = M_D \bullet M_A$, implying that M_B is a solution for the series topology of the equation $M_X \bullet M_A \preceq M_C$. To argue that $M_B \bullet M_A = M_D \bullet M_A$, we prove first that $\lambda_{B \bullet A} = \lambda_{D \bullet A}$ and next that $\delta_{B \bullet A} = \delta_{D \bullet A}$ on the appropriate domains of definition.

First we show that for each state $((s, s, s_A), s_A) \in ((S, S, S_A), S_A)$ of the series composition $M_B \bullet M_A$ and $(s, s_A) \in (S, S_A)$ of the controller's composition $M_D \bullet M_A$, and for each input $i \in I$

$$\lambda_{B \bullet A}(((s, s, s_A), s_A), i) = \lambda_{D \bullet A}((s, s_A), i).$$

For each state $((s, s, s_A), s_A) \in ((S, S, S_A), S_A)$ of the series composition $M_B \bullet M_A$ and for each input $i \in I$, by definition of series composition, it holds

$$\lambda_{B \bullet A}(((s, s, s_A), s_A), i) = \lambda_A(s_A, \lambda_B((s, s, s_A), i))$$

and, by definition of λ_B,

$$\lambda_A(s_A, \lambda_B((s, s, s_A), i)) = \lambda_A(s_A, \lambda_D(s, (i, \lambda_{D \bullet A}((s, s_A), i)))).$$

Furthermore, by definition of controller's composition,

$$\lambda_{D \bullet A}((s, s_A), i) \subseteq \lambda_A(s_A, \lambda_D(s, (i, \lambda_{D \bullet A}((s, s_A), i)))),$$

where the containment relation can be replaced by strict equality under the hypothesis that A is a DFSM [1], yielding

$$\lambda_{D \bullet A}((s, s_A), i) = \lambda_A(s_A, \lambda_D(s, (i, \lambda_{D \bullet A}((s, s_A), i)))).$$

Putting it all together

$$\lambda_{B \bullet A}(((s, s, s_A), s_A), i) = \lambda_A(s_A, \lambda_D(s, (i, \lambda_{D \bullet A}((s, s_A), i)))) = \lambda_{D \bullet A}((s, s_A), i).$$

Next we show that for each state $((s, s, s_A), s_A) \in ((S, S, S_A), S_A)$ of the series composition $M_B \bullet M_A$ and $(s, s_A) \in (S, S_A)$ of the controller's composition $M_D \bullet M_A$, and for each input $i \in I$

$$\delta_{B \bullet A}(((s, s, s_A), s_A), i) = ((s', s', s'_A), s'_A)$$

[1] By definition of series composition it is always $\lambda_{B \bullet A}(((s, s, s_A), s_A), i) = \lambda_A(s_A, \lambda_B((s, s, s_A), i))$, whereas for controller's composition $\lambda_{D \bullet A}((s, s_A), i) \subseteq \lambda_A(s_A, \lambda_D(s, (i, \lambda_{D \bullet A}((s, s_A), i))))$, because an output $\tilde{o}$, issued at state s_A of M_A under input v, may be "blocked", meaning that from state s of M_D there is no transition with label $i\tilde{o}/v$. The hypothesis that M_A is a DFSM is sufficient (but not necessary) to guarantee that the previous situation cannot happen and so the containment relation becomes a strict equality. .

where

$$(s', s'_A) = \delta_{D \bullet A}((s, s_A), i).$$

Indeed, if $\delta_{D \bullet A}((s, s_A), i) = (s', s'_A)$, then $\delta_D(s, (i, \lambda_{D \bullet A}((s, s_A), i))) = s'$ and, by definition of δ_B, it is $\delta_B((s, s, s_A), i) = (s', s', s'_A)$. Finally, from $\delta_A(s_A, \lambda_D(s, (i, \lambda_{D \bullet A}((s, s_A), i)))) = s'_A$, it follows $\delta_{B \bullet A}(((s, s, s_A), s_A), i) = ((s', s', s'_A), s'_A)$.

The equalities $\lambda_{B \bullet A} = \lambda_{D \bullet A}$ and $\delta_{B \bullet A} = \delta_{D \bullet A}$ hold for the initial states and inductively for the correspondingly reached states, implying that $M_B \bullet M_A$ and $M_D \bullet M_A$ are equivalent. □

Corollary 2.1 *Given a plant $M_A = (S_A, V, O, T_A, r_A)$ and a reference model $M_C = (S_C, I, O, T_C, r_C)$, if M_A is a DFSM then the FSM equation $M_X \bullet M_A \preceq M_C$ has a complete solution for the series topology with an unknown head component iff it has a complete solution for the controller's topology.*

Say that a solution M_D is compositionally complete if the composition $M_D \bullet M_A$ is a complete FSM. Under the same hypothesis, the FSM equation $M_X \bullet M_A \preceq M_C$ has a compositionally complete solution for the series topology with an unknown head component iff it has a compositionally complete solution for the controller's topology.

Proof. The construction in the proof of Th. 2.1 establishes immediately both results. □

3 Conclusions

Given the problem of finding an unknown component that combined with a known part of a system satisfies a desired specification, we have been developing a solution technique based on defining equations over languages associated to the components of the system. We introduced two composition operators for abstract languages: synchronous composition, $\bullet$, and parallel composition, $\diamond$, and we studied the most general solutions of the language equations $A \bullet X \subseteq C$ and $A \diamond X \subseteq C$ ($\subseteq$ denotes language containment), defining the language operators required to express them. In particular we studied the solutions of the equations defined over finite state machines (FSMs) of the type $M_A \bullet M_X \subseteq M_C$ and $M_A \diamond M_X \subseteq M_C$, where M_A models the context, M_C models the specification and M_X is unknown.

Investigating the solutions for various network topologies of synchronous FSM equations, we have been led to the question on the relation between the solvability of an equation for the series topology and the corresponding equation for the controller's topology. In this note we answer to the question, establishing that, if M_A is a deterministic FSM, then the FSM equation $M_X \bullet M_A \preceq M_C$ is solvable for the series topology with an unknown head component iff it is solvable for the controller's topology. Our proof is constructive, meaning that given a solution M_B for the series topology shows how to build a solution M_D for the controller's topology and viceversa.

A systematic effort to outline a complete theory of FSM equations is under way and we will report on it soon. We plan also to implement a synthesis system based on solving language equations.

4 Acknowledgments

The first author was partly supported by the Russian Fund of Basic Research (Grant 99-01-00337). 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] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

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

[3] 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.

[4] N. Yevtushenko, T. Villa, R. Brayton, A. Petrenko, and A. Sangiovanni-Vincentelli. Sequential synthesis by language equation solving. *The Proceedings of the International Workshop on Logic Synthesis*, June 2000.

[5] 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.

A Equations over Languages

A.1 Languages

It is useful to remind the notions of substitution and homomorphism of languages [1]. A **substitution** f is a mapping of an alphabet Σ onto subsets of $\Delta^\star$ for some alphabet Δ. The substitution f is extended to strings by setting $f(\epsilon) = \epsilon$ and $f(xa) = f(x)f(a)$. An **homomorphism** h is a substitution such that $h(a)$ is a single string for each symbol a in the alphabet Σ.

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

   $$h((x, v)) = x, \ \forall x \in X, \forall v \in V,$$

   then the language

   $$L_{\downarrow X} = \{h(\alpha) \mid \alpha \in L\}$$

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

2. Given a language L over alphabet X and an alphabet V, consider the substitution f defined as

   $$f(x) = (x, v), \ \forall x \in X, \forall v \in V,$$

   then the language

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

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

We introduce some useful classes of languages.

Definition A.1 *A language L over alphabet X is* **prefix-closed** *if* $\forall \alpha \in X^\star \ \forall x \in X \ [\alpha x \in L \Rightarrow \alpha \in L]$. *Equivalently, L is prefix-closed iff $L = Init(L)$.*

Definition A.2 *A language L over alphabet $X = I \times O$ is I*-**progressive** *if* $\forall \alpha \in X^\star \ \forall i \in I \ \exists o \in O \ [\alpha \in L \Rightarrow \alpha \ (i, o) \in L]$.

A.2 Synchronous Composition of Languages

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

Definition A.3 *Given alphabets I, U, O, language L_1 over $I \times U$ and language L_2 over $U \times O$, the* **synchronous composition** *of languages L_1 and L_2 is the language* [2] $[(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}]_{\downarrow I \times O}$*, denoted by $L_1 \bullet L_2$, defined over $I \times O$.*

A.3 Solution of Language Equations under Synchronous Composition

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

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

Definition A.4 *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 $B \neq \emptyset$ and $A \bullet B \subseteq C$. The* **largest solution** *is the solution that contains any other solution.*

Theorem A.1 *The largest solution of the equation $A \bullet X \subseteq C$ is the language $S = \overline{A \bullet \overline{C}}$, if $S \neq \emptyset$.*

Proof. Consider a string $\alpha \in (U \times O)^\star$, 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:

$$\begin{aligned} A \bullet \{\alpha\} \subseteq C \quad &\Leftrightarrow \\ (A_{\uparrow O} \cap \{\alpha\}_{\uparrow I})_{\downarrow I \times O} \cap \overline{C} = \emptyset \quad &\Leftrightarrow \\ A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U} = \emptyset \quad &\Leftrightarrow \\ \alpha \notin (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O} \quad &\Leftrightarrow \\ \alpha \in \overline{(A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U \times O}} \quad &\Leftrightarrow \\ \alpha \in \overline{A \bullet \overline{C}} \quad & \end{aligned}$$

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

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

if $S \neq \emptyset$. □

Corollary A.1 *A language $B \neq \emptyset$ over alphabet $U \times O$ is a solution of $A \bullet X \subseteq C$ iff $B \subseteq \overline{A \bullet \overline{C}}$. If $\overline{A \bullet \overline{C}} = \emptyset$, then the language equation $A \bullet X \subseteq C$ has no solution.*

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., of being prefix-closed, progressive, etc.

If S is prefix-closed then S is the largest prefix-closed solution of the equation. However, not each 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.

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

Proposition A.1 *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 it holds $\alpha(u, o) \in S$.

Proposition A.2 *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.

B Equations over Finite State Machines

B.1 Finite State Machines

Definition B.1 *A* **non-deterministic FSM** *(NDFSM), or simply an* **FSM** *or a* **machine***, is defined as 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 NDFSM at present state p may transit to next state n and output o if and only if $(i, p, n, o) \in T$. State $r \in S$ represents the initial or reset state. We denote the restriction of 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 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 a 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 a present state and input pair, the FSM is said to be* **partial***. An FSM is said to be* **trivial** *when $T = \emptyset$.*

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

Definition B.2 *A* **deterministic FSM** *(DFSM) is an NDFSM 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.*

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 $o = T_o(i, p)$.

B.2 Languages of FSMs

We now introduce the notion of language associated to an FSM. This is achieved by looking to the automaton underlying a given FSM. For our purposes, we define an associated language over the alphabet $I \times O$.

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).

Definition B.3 *Given an FSM $M = \langle S, I, O, T, r\rangle$, consider the finite automaton $F(M) = \langle S, I \times O, \Delta, r, S\rangle$, where $((i, o), s, s') \in \Delta$ iff $(i, s, s', o) \in T$. The language accepted by $F(M)$ is denoted $L_r^{\times}(M)$, and by definition is the* $\times$**-language** *of M at state r. Similarly $L_s^{\times}(M)$ denotes the language accepted by $F(M)$ when started at state s, and by definition is the $\times$-language of M at state s.*

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

Definition B.4 *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_A has property X if its associated FSM language has property X.

Definition B.5 *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 supposed to have the same input 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 $L_t(M_B) = L_s(M_A)$. States that are not equivalent are called* **distinguishable***. An FSM whose states are all pairwise distinguishable 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., $M_A \cong M_B$. Machines that are not equivalent are* **distinguishable***.*

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

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

Theorem B.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, from which it is possible an extension under any input.

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 language of an FSM over input alphabet I and output alphabet O.

Procedure B.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 B.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}.

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

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 B.2 *Procedure B.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, $k > 1$. 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 B.1 *An FSM whose language is L^{FSM} or $Prog(L^{FSM})$ can be trivially deduced from $\hat{A}$ by interpreting each label (i, o) as an input/output pair i/o.*

Proposition B.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 $L(M)$ of M is contained in L if and only if $L(M) \subseteq L^{FSM}$.*

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

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

B.3 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 $O_1 \times O_2 = O$.

We remind that, by definition of synchronous composition of languages, a sequence $\alpha \in (I_1 \times I_2 \times O_1 \times O_2)^\star$ 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 B.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. $\square$

Therefore we can state the following definition.

Definition B.6 *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.

The previous definition is sound because the language $L(M_A) \bullet L(M_B)$ by Lemma B.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_B and M_A to the automata accepting their FSM languages, operate on them and then we convert back from the resulting automaton to an FSM; then we produce a reduced PNDFSM (we assume that M_B and M_A are PNDFSMs).

B.4 FSM Equations under Synchronous Composition

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 $O_1 \times O_2$, let us consider the FSM equation

$$M_A \bullet M_X \preceq M_C, \tag{3}$$

whose unknown is an FSM M_X over inputs $I_2 \times U$ and outputs $V \times O_2$. Note that sometimes we will use the shortened notation $I = I_1 \times I_2$ and $O = O_1 \times O_2$.

Definition B.7 *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, if and only if $M_A \bullet M_B \preceq M_C$.*

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

$$L(M_A) \bullet L(M_X) \subseteq L(M_C), \tag{4}$$

[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 $L(M_B)_{\uparrow I_1 \times O_1}$.

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

$$A \bullet X \subseteq C. \tag{5}$$

We want to characterize the solutions of $A \bullet X \subseteq C$ as FSM languages. We know from Theorem A.1 in Sec. A that the largest solution of the equation $A \bullet X \subseteq C$ is the language $S = \overline{A \bullet \overline{C}}$, if $S \neq \emptyset$.

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

Theorem B.3 $S \neq \emptyset$.

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., $\epsilon \in S \neq \emptyset$. $\square$

Thus, a synchronous FSM language equation is always solvable, since the solution includes at least the empty string, that corresponds to the trivial FSM being always a solution of a synchronous FSM equation.

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 B.4 *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 B.1 to S.* S^{FSM} *contains at least the string* ϵ*.*

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 B.2 to S^{FSM}, yielding $Prog(S^{FSM})$, the largest $(I_2 \times U)$-progressive FSM language $\subseteq (I_2 \times U \times V \times O)^\star$. Notice that an $(I_2 \times U)$-progressive solution might not exist, and in that case Procedure B.2 returns an empty language.

Proposition B.3 *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, if and only if* M_B *is a reduction of the FSM* M_S *associated to* S^{FSM}*, where* S^{FSM} *is obtained by applying Procedure B.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 Procedure B.2. A complete FSM is a solution if and only if it is a reduction of the largest complete solution* $M_{Prog(S^{FSM})}$*.*