Automated Synthesis of Protocol Converters with BALM-II

Giovanni Castagnetti[1], Matteo Piccolo[1], Tiziano Villa[1](✉), Nina Yevtushenko[2], Robert Brayton[3], and Alan Mishchenko[3]

[1] Dipartimento d'Informatica, Università di Verona, Verona, Italy
tiziano.villa@univr.it
[2] Computer Science Laboratory, Tomsk State University, Tomsk, Russia
[3] Department of EECS, University of California, Berkeley, CA, USA

Abstract. We address the problem of the automatic design of automata to translate between different protocols, and we reduce it to the solution of equations defined over regular languages and finite automata (FA)/finite state machines (FSMs). The largest solution of the defined language equations includes all protocol converters that solve the problem; this is a strong advantage over computational techniques that deliver only one or a few solutions, which might lead to suboptimal implementations (e.g., as sequential circuits). Our model is versatile, because it can handle different topologies and constraints on the solutions. We propose a fully automatic procedure implemented inside a software package BALM-II which solves language equations. For illustration we show examples of setting up and solving language equations for classical protocol mismatch problems, aiming at the design of protocol converters to interface an alternating-bit (AB) sender and a non-sequenced (NS) receiver. Our automatic converter synthesis procedure yields a complete solution for automata and FSMs, and may serve as a core engine to embed into any full-fledged interface synthesis tool.

1 Introduction

In electronic system level design, an important step is the implementation of communication, e.g., consider a design scenario based on the composition of reusable Intellectual Property (IP) cores that realize different communication protocols (as in [1]). This problem may be addressed either by interface template instantiation (customization of a set of templates from functional and performance requirements), or by interface synthesis [2]. The latter aims to adapt incompatible transmitters and receivers of data, by designing a converter (if it exists) that makes them compatible within the given constraints of the implementation.

For instance, consider the example (from [2,3]) of a producer and a consumer communicating complex data each partitioned into two parts. The producer (handshake protocol) can wait an unbounded amount of time between sending the two parts, whereas the consumer (serial protocol) must receive the second part just after the first one with no intermediate wait. The further specification is

D. Bianculli et al. (Eds.): SEFM 2015 Workshops, LNCS 9509, pp. 281–296, 2015.
DOI: 10.1007/978-3-662-49224-6_23

that data - to be transmitted in the same order - are neither lost nor duplicated. The goal is to synthesize a converter that takes the first part of a datum sent by the producer and delivers it to the consumer, only after it receives also the second part of the datum.

In this paper we address automatic interface synthesis when the protocols and specification are expressed by automata and regular languages, and obey an FSM semantics to be specified according to the composition operators: given protocols P and Q, synthesize the most general converter X such that the composition of P, Q and X behaves like a required specification C.

We model the problem of synthesizing a converter as follows: given a module A (the context, i..e, the product of the two protocols P and Q) and a specification module C, synthesize the unknown converter X such that the composition of A and X conforms to or refines C. We allow any communication topology between the protocols and the converter.

The main contributions of this paper are:

1. We reduce the problem to solving inequalities and equations over regular languages/automata (FA) and finite state machines (FSMs), for which there are closed-form solutions, with respect to the synchronous and parallel composition operators (see [4–9]).
2. We describe a new package - BALM-II (see [10,11]) - for solving inequalities and equations over FA/FSMs, which we developed to extend to equations over parallel composition the previous software package BALM (see [12]) that was restricted to equations over synchronous composition.

In Sect. 2 we survey briefly the previous work, whereas in Sect. 3 we outline the theory and practice of solving inequalities and equations over FA/FSMs with BALM-II, a software package that solves synchronous and parallel equations. In Sect. 4 we apply this synthesis technique to an example of protocol mismatch problem modeled with parallel equations. We conclude in Sect. 5 by summarizing a comparison with other approaches and mentioning future work.

2 Previous Work

The problem of protocol converter synthesis received recently a lot of attention in the literature (for instance, see the references [1,2,13–23]).

In [15–18], Passerone *et al.* considered as composition any binary operator that satisfies appropriate algebraic properties; the proposed definition includes parallel and synchronous composition of automata as special cases (see Sect. 3 for their formal definition). Moreover, they considered various preorder conformance relations of which containment is the simplest case (it is only required that the composition is monotonic). To synthesize a converter, they introduced a mirror function that is not unique for an agent, and instead satisfies appropriate properties depending on the composition operator and conformance relation.

This general setting reduces to our model when protocols are described by automata. For instance, for parallel inequalities/equations over automata with respect to the containment relation, the complement operator appearing in the closed-form $X = \overline{A \diamond \overline{C}}$ is exactly the mirror function, and so both approaches capture all solutions.

Further requirements on the synthesized converter are mentioned in [17]: one is the extraction of a deterministic solution for the serial topology (for which it is known that selecting a complete submachine is sufficient), and the other is about enforcing progressiveness. About the latter, defined as the property that each action defined by the specification should be provided by the composition, it is suggested to enforce it by deleting transitions from the largest solution; however, there is no discussion that in general (in the rectification topology, for example) this trimming operation is sufficient only when the largest solution is represented by a perfect automaton (see [24,25] for details).

Jiang and Jin studied in [14] a variant where the composition of the given protocols and of the converter behaves like a FIFO with a buffer of size k.

In [20], Bhaduri and Ramesh solved the problem of converter synthesis for interface automata with respect to an alternating simulation relation, i.e., a relation ρ from P to Q for which $(s, t) \in \rho$ implies that all input moves from t can be simulated by s and all output moves from s can be simulated by t. A refinement between interface automata is defined as the existence of an alternating simulation between the initial states, i.e., an interface automaton P refines an interface automaton Q, $P \preceq Q$, if (1) the set of input actions of Q is a subset of the input actions of P, (2) the set of output actions of P is a subset of the output actions of Q, and (3) there is an alternating simulation ρ from P to Q such that $(s_0, t_0) \in \rho$, where s_0 and t_0 are initial states, respectively, of P and Q. The most abstract solution under alternating simulation of $P \parallel R \preceq Q$ is given by $R = (P \parallel Q^{\perp})^{\perp}$, where $P^{\perp}$ is the same as P, except that the input actions in P become the output actions in $P^{\perp}$ and similarly the output actions in P are the input actions in $P^{\perp}$. This $P^{\perp}$ operator reminds of the inverse FSM introduced in the solution of the model matching problem discussed in [26].

In [21] Avnit and Sowmya discussed a variant of the converter synthesis problem, in which, instead of requiring that the composition satisfies the specification, the objective is to insure what is called a correct conversion: a correct converter is compatible with both protocols, and never causes an overflow or an underflow of its buffers. The problem is solved by deleting redundant transitions; the authors do not discuss the fact that when dealing with sequential systems one should rather delete redundant sequences than redundant transitions, otherwise, some solutions may be lost.

Given two automata representing two protocols, in [19] Watanabe *et al.* introduced a synthesizer that judges whether each state of the product automaton is legal or not in terms of data dependency. The output transducer is an FSM which is the subset of the product automaton that consists of legal states. A transducer is required to satisfy the property that illegal tuples are removed according to the following rules: (a) a data word which has not been received must not be sent; (b) if a state from the master automaton is a final state, all the states from the slave automaton must be final states; (c) tuples which inevitably transit to an illegal tuple are also illegal. The first rule does not care about which automaton has received or sent a data word. The resulting solution is a submachine of the product automaton, i.e., all machines have the same set of actions. There is no explicit mention of the overall specification, similar to a case-study in [17], where the transducer is a given. Notice that each component protocol is represented as a composition of smaller automata, so that there is no need of a monolithic component automaton when deriving a transducer. In summary, they construct the whole transducer from a set of partial transducers, and the synthesized transducer consists of several FSMs to handle parallel transactions.

Cao and Nymeyer presented in [27] a theoretical model of a converter including buffers and allowing the specification by the designer of CTL conditions, to be verified by means of a model checker.

Sinha et al. developed in [1,23,28–30] a compositional approach for the integration of multiple components with a wide range of protocol mismatches into a single System-on-Chip (SoC). They construct the SoC either in a single step or incrementally, and discuss the pros and cons of the two approaches; the latter has advantages with respect to scalability, reuse, wiring congestion and latency errors. They can handle mismatches such as multiple clocks, multi-directional IP communication, control and data-width mismatches, but not yet complex mismatches such as interface inconsistencies. They also ensure the satisfaction of multiple constraints (specifications) on the behaviour of the SoC, which are specified as properties in temporal logic CTL; these properties are classified as control constraints over the state labels in the protocols, data constraints over the data counters used to track down the data communication between IPs, and control-data constraints using state labels and counter values in the same CTL formula. Protocols are described as Synchronous Kripke Structures (SKS) which are Finite State Machines (FSMs), augmented with a set of propositions (denoting control labels and data labels) that label each state to describe its control and data input/output status. In the proposed methodology, a clock automaton is introduced to handle multiple clocks, i.e., each protocol is oversampled to describe its behaviour with respect to the fastest clock in the SoC. Then the parallel composition of all the SKSs is computed, and finally inputs and outputs are partitioned as follows: uncontrollable signals passed to/from the protocols as soon as available, shared signals emitted and read by protocols, missing control signals to be generated by the converter. The synthesis of converters is based on a recursive algorithm originally presented in [31], which is extended to construct a graph corresponding to *all* the valid behaviours of the protocols such that the given constraints are satisfied (the original algorithm did not enumerate all possible ways in which an assertion may be satisfied). Then a deterministic converter is obtained by extracting a sub-graph from the previous graph representing the maximal non-deterministic converter. The converter behaves acts as follows: it forwards all uncontrollable signals to the environment/protocols in the same clock tick; it buffers shared signals from the protocols and forwards them after one or more clock ticks; it maintains a 1-place buffer for each shared signal; it may hide a buffered signal from the protocols in a tick to disable an undesirable transition, and it may generate some control signals emitted neither by the environment nor by the protocols. The converter executes based on the fastest clock of the SoC, and at each step it follows a precise sequence of micro-steps with the environment and the protocols: it reads the uncontrollable signals from the environment as an input formula, it forwards the uncontrollable signals and converter-controlled signals (buffered or generated) to the protocols, it reads the signals generated by the IPs emitting any uncontrollable outputs to the environment and buffering relevant control signals. All together these actions form a macro-step; the result is a converter SKS (CSKS), whose parallel composition with the protocols defines the matched system. Results are reported with SKS abstractions of IP protocols extracted from Advanced Microcontroller Bus Architecture (AMBA) white papers and Hardware Description Language (HDL) implementations.

In [32] Autili et al. survey the automatic synthesis of connectors at the level of networked systems to achieve protocol interoperability, i.e., the ability to communicate and coordinate correctly. The challenge is for heterogeneous protocols in an ubiquitous computing environment to cooperate to reach some common goals, even though meeting dynamically and without a priori knowledge of each other. Connectors may be either coordinators (the networked systems are already able to communicate, but they need to be coordinated) or mediators (the requirement is both to enable communication and achieve coordination). Even though automated and run-time interoperability is still an open challenge for networked systems, the paper reports the state-of-art on formal methods for the automatic synthesis of coordinators and mediators, concluding with necessary and sufficient interoperability conditions to guarantee the existence of correct mediators. Automatic synthesis of a secure orchestrator for a set of BPMN (Business Process Model and Notation) processes is presented in [33], based on synthesis by partial model checking.

In the following sections, we will present a theoretical approach and a software tool to synthesize all protocol converters for any topology of parallel and synchronous composition, when the components to interface are modeled by finite automata and finite state machines; we will compare in Sect. 5 with the most relevant previous work.

3 Equations over Languages and Automata

Consider a composition topology, shown in Fig. 1, where the composition of two interconnected components (the context or plant A, and the unknown component X) defines a behaviour contained or equal to the one defined by the specification C; the context and the unknown interact through internal signals, and communicate with the environment with external input and output signals. According to what connections are present, one may define simplified topologies, e.g., the rectification topology when the unknown component has no external input and output signals. Our goal is to find the most general unknown component that contains all the behaviours such that the composition is included in the specification. In the next subsections, we will introduce two types of composition:

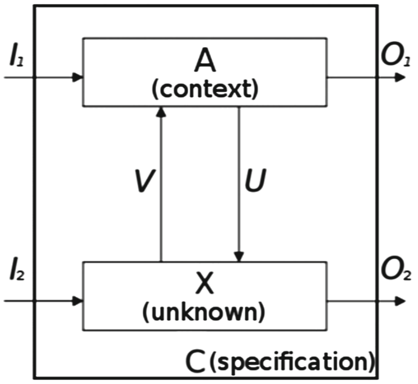

Fig. 1. General topology.

parallel composition (*a.k.a.* as **asynchronous composition**) and **synchronous composition**.

3.1 Equations with Parallel Composition

In order to introduce parallel composition we must define two new operators: expansion and restriction. The first one, denoted by the symbol $\Uparrow$, is used to extend the alphabet of the automaton, while the second one, denoted by the symbol $\Downarrow$, is used to restrict the alphabet of the automaton.

Definition 1. *Given a language L over alphabet X and an alphabet V, consider the mapping $e : X \to 2^{(X \cup V)^\star}$ defined as $e(x) = \{\alpha x \beta \mid \alpha, \beta \in (V \setminus X)^\star\}$, 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 $(V \setminus X)^\star$. Notice that $e(\epsilon) = \{\alpha \mid \alpha \in (V \setminus X)^\star\}$.*

Definition 2. *Given a language L over alphabet $X \cup V$, consider the homomorphism $r : X \cup V \to V^\star$ defined as $r(y) = y$ if $y \in V, r(y) = \epsilon$ if $y \in X \setminus V$, 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 in V. Notice that $r(\epsilon) = \epsilon$.*

Definition 3. *Given the pairwise 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 $L_1 \diamond_{I \cup O} L_2$, defined over $I \cup O$.*

With these operators it is possible to define an equation over languages and write a closed form for its solution.

Definition 4. *Given the 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 $A \diamond B \subseteq C$.*

Definition 5. *The* **largest solution** *is a solution that contains any other solution.*

Theorem 1. *(proof in [9], Theorem 2.16, p. 24) The largest solution of the equation $A \diamond X \subseteq C$ is the language $S = \overline{A \diamond \overline{C}}$.*

Language equations can be solved effectively when their languages have finitary representations, usually by means of automata generating them, e.g., finite automata for regular languages. In that case, we can interpret a given language equation as denoting an equation over the automata generating the given languages. So, we talk interchangeably of language equation or automaton equation (where the automata generate the languages).

Definition 6. *A* **finite automaton** *(FA) is a 5-tuple $F = \langle S, \Sigma, \Delta, r, Q \rangle$. S represents the finite state space, Σ represents the finite alphabet of actions, and $\Delta \subseteq \Sigma \times S \times S$ is the next state relation, such that $(\sigma, p, n) \in \Delta$ iff $n \in S$ is a next state of present state $p \in S$ on action $i \in \Sigma$. The initial or reset state is $r \in S$ and $Q \subseteq S$ is the set of final or accepting states. The automaton F is deterministic, if for each state $s \in S$ and any action $\sigma \in \Sigma$ there exists at most one state $s\prime$, such that $(\sigma, s, s\prime) \in \Delta$, otherwise it is nondeterministic. A variant of FA allows the introduction of ϵ-moves, meaning that $\Delta \subseteq (\Sigma \cup \{\epsilon)\} \times S \times S$; by the closure procedure one obtains an equivalent deterministic FA without ε-moves [34].*

Finally, we introduce equations over FSMs, by interpreting them as equations over languages produced by FSMs; these languages are regular and so are generated by finite automata.

Definition 7. *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.*

For ease of discussion, we assume that FSMs are complete, i.e., at least one transition is specified for each present state and input pair.

There are different ways of associating a language to an FSM, by considering the automaton underlying the FSM, according to the semantics of choice. Here we introduce an interleaving semantics, naturally associated with parallel composition, where the language of an FSM is defined over the alphabet $I \cup O$. as follows.

Definition 8. *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 $L_r^{\cup}(M)$, and by* *definition is the* $\cup$-**language** *of* M *at state* r*. Similarly* $L_s^{\cup}(M)$ *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,* $L_s^{\cup}(M) \subseteq (IO)^{\star}$*, where* IO *denotes the set* $\{io \mid i \in I, o \in O\}$*.*

In short, 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. This automaton generates the language that we associate to the FSM, and it can be interpreted as an Input/Output Automaton [35].

Similarly, we can define the composition of FSMs and equations over FSMs.

Definition 9. *The* **parallel composition of FSMs** M_A *and* M_B *yields a reduced observable FSM* $M_A \diamond M_B$ *with language*

$$L(M_A \diamond M_B) = L(M_A) \diamond L(M_B) \cap \ (IO)^{\star}.$$

The previous definition relies on the fact that $L(M_A \diamond M_B) = L(M_A) \diamond L(M_B) \cap (IO)^{\star}$ is a unique FSM language to which corresponds any FSM whose associated automaton accepts such language (there are many such behaviorally equivalent FSMs). The resulting FSM may be made unique by standard operations like subset construction and state minimization, which produce a reduced observable FSM. By definition, an FSM is observable if for each triple $(i, p.o) \in I \times S \times O$ there is at most one next state n such that $(i, p, n, o) \in T$, i.e., for each input, present state and output there is a unique next state, equivalent to the fact that the underlying finite automaton is deterministic; an FSM is reduced if there are no two equivalent states, i.e., states that cannot be distinguished by their external behaviour.

Definition 10. *The* **parallel FSM equation**

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

corresponds to the related language equation

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

where $L(M_A)$ *and* $L(M_C)$ *are the FSM languages associated with FSMs* M_A *and* M_C*.*

In the following, we summarize the steps to solve parallel FSM equations. For ease of notation, given an FSM M_X we denote by X its associated automaton and generated language.

Given the parallel FSM equation

$$M_A \diamond_{I_1 \cup I_2 \cup O_1 \cup O_2} M_{X_{I_2 \cup U \cup V \cup O_2}} \subseteq M_C, \tag{1}$$

one derives the corresponding automata A and C and solves the automaton equation

$$A \diamond_{I_1 \cup I_2 \cup O_1 \cup O_2} X_{I_2 \cup U \cup V \cup O_2} \subseteq C, \tag{2}$$

equivalent to

$$((A_{I_1 \cup V \cup U \cup O_1})_{\Uparrow I_2 \cup O_2} \cap (X_{I_2 \cup U \cup V \cup O_2})_{\Uparrow I_1 \cup O_1})_{\Downarrow I_1 \cup I_2 \cup O_1 \cup O_2} \subseteq C_{I_1 \cup I_2 \cup O_1 \cup O_2}.$$

The largest automaton solution is given by

$$X = \overline{A \diamond_{I_2 \cup U \cup V \cup O_2} \overline{C}} \tag{3}$$

equivalent to

$$X_{I_2 \cup U \cup V \cup O_2} = \overline{((A_{I_1 \cup V \cup U \cup O_1})_{\Uparrow I_2 \cup O_2} \cap (\overline{C_{I_1 \cup I_2 \cup O_1 \cup O_2}})_{\Uparrow U \cup V})_{\Downarrow I_2 \cup U \cup V \cup O_2}}. \tag{4}$$

The Eq. (3) is intuitively derived as follows: we take the unwanted behaviours (the complement of the specification) and we compose them with the context in order to obtain only the behaviours that we want to delete from our system. Then we complement this result and so we obtain all the acceptable behaviours; by construction this is the largest automaton solution of the Eq. (2). In a similar it way is possible to obtain the largest FSM solution of the Eq. (1), as follows.

The largest FSM solution requires enforcing the hypothesis that an input must be followed by an output before another input can be produced. This particular constraint is necessary because we do not assume any extension of the FSM model to store symbols (i.e., FSMs with buffers).

Setting $I = I_1 \cup I_2$ and $O = O_1 \cup O_2$, the largest FSM solution is given by

$$X = \overline{A \diamond_{I_2 \cup U \cup V \cup O_2} (\overline{C} \cap (IO)^\star)} \cap (UV)^\star \tag{5}$$

that is equivalent to

$$\begin{aligned} X_{I_2 \cup U \cup V \cup O_2} = {} & \overline{((A_{I_1 \cup V \cup U \cup O_1})_{\Uparrow I_2 \cup O_2} \cap (\overline{C_{I_1 \cup I_2 \cup O_1 \cup O_2}} \cap (IO)^\star)_{\Uparrow U \cup V})_{\Downarrow I_2 \cup U \cup V \cup O_2}} \\ & \cap ((UV)^\star)_{\Uparrow I_2 \cup O_2}. \end{aligned} \tag{6}$$

Once the unknown X has been computed, as a sanity check one may verify whether the computed X satisfies the inequality (2)

$$A \diamond_{I_1 \cup I_2 \cup O_1 \cup O_2} X_{I_2 \cup U \cup V \cup O_2} \subseteq C,$$

equivalent to

$$((A_{I_1 \cup U \cup V \cup O_1})_{\Uparrow I_2 \cup O_2} \cap (X_{I_2 \cup U \cup V \cup O_2})_{\Uparrow I_1 \cup O_1})_{\Downarrow I_1 \cup I_2 \cup O_1 \cup O_2} \subseteq C_{I_1 \cup I_2 \cup O_1 \cup O_2}. \tag{7}$$

If the composition is equivalent to the specification then the equation is solvable, otherwise a solution of the inequality yields a strict refinement of the specification. Similar formulas can be derived for the simplified topology or other topologies, introducing the appropriate supports of variables.

3.2 Equations with Synchronous Composition

We can repeat the previous steps to define equations with respect to synchronous composition and obtain dual results in terms of characterization of the solutions. To save space, we will not go through again the all derivation, but only point out the main differences, and refer to [9] for the details.

To define synchronous composition, instead of the operators of expansion and restriction, we need lifting and projection.

Definition 11. *Given a language L over alphabet $X \times V$, consider the homomorphism $p : X \times V \to V^\star$ defined as $p((x, v)) = v$, then the language $L_{\downarrow V} = \{p(\alpha) \mid \alpha \in L\}$ over alphabet V is the* **projection** *of language L to alphabet V, or V-projection of L. By definition of substitution $p(\epsilon) = \epsilon$.*

Definition 12. *Given a language L over alphabet X and an alphabet V, consider the substitution $l : X \to 2^{(X \times V)^\star}$ defined as $l(x) = \{(x, v) \mid v \in V\}$, then the language $L_{\uparrow V} = \{l(\alpha) \mid \alpha \in L\}$ over alphabet $X \times V$ is the* **lifting** *of language L to alphabet V, or V-lifting of L. By definition of substitution $l(\epsilon) = \{\epsilon\}$.*

The given substitution operators change a language and its alphabet of definition; in particular the operators $\uparrow$ and $\downarrow$ vary the components that are present in the Cartesian product defining the language alphabet.

Definition 13. *Given the alphabets I, U, O, language L_1 over $I \times U$ and language L_2 over $U \times O$, the* **synchronous composition** *of languages L_1 and L_2 is the language $[(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}]_{\downarrow I \times O}$, denoted by $L_1 \bullet_{I \times O} L_2$, defined over $I \times O$.*

Finally, the language associated to an FSM under the semantics of synchronous composition is the one generated by the automaton coinciding with the original FSM where all states are made accepting and the edges carry a label of the type (i, o).

Definition 14. *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* **×-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.*

The rest follows by mimicking the steps described in Sect. 3.1.

3.3 BALM-II

All these operations are available in the new software package BALM-II [11], which is an extension of an existent software called BALM [12].

BALM, a branch of the MVSIS [36] project, implements many operations to solve synchronous inequalities and equations over automata and FSMs, like lifting and projection, which can be used in the synthesis/resynthesis of sequential circuits. However, the original version of BALM did not handle inequalities and equations with respect to parallel composition; therefore BALM was upgraded to BALM-II, a version extended with new procedures and commands to solve parallel equations (see the User's manual in [10]).

BALM-II inherits all the representation formats and commands from BALM, and extends it with features to represent and manipulate automata encoding FSMs interpreted with the semantics of parallel composition, (i.e., expansion, restriction, etc.). Notice that, under the semantics of synchronous composition, FSMs are translated into automata simply by merging the input and output variables of the FSM into the "inputs" of the automaton, since the automata are defined over the cartesian product $A_1 \times A_2 \times \ldots \times A_n$, where $A_1, \ldots, A_n$ are the FSM's alphabets; instead, under the semantics of parallel composition, the translation is less straightforward, and it requires splitting the transitions of the FSM and introducing new states, because the resulting automata are defined over the union of alphabets $A_1 \cup A_2 \cup \ldots \cup A_n$ (see [9,10] for a description of the transformation procedure). BALM-II makes available the commands to translate FSMs into the corresponding automata, then to operate on the automata to solve the equations, and finally to extract from the automata the resulting FSMs (of course, if the original problem is specified directly by means of automata, there is no need of this round trip from and to FSMs). This can be done for both the synchronous and parallel semantics. We refer to [9] for a theoretical exposition, and to [10] for implementation details and examples of usage.

In the next section we will apply this computational framework to the synthesis of the largest protocol converter on case-studies from the literature. For simplicity, we will describe examples modeled directly by automata, and refer the reader to [10] for examples with FSMs showing also the conversion procedures from FSMs to automata and viceversa.

4 An Example of Asynchronous Protocol

We define and solve an equation over finite automata to solve a problem of converter synthesis, i.e., the design of an automaton translating between two different protocols. 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 behavior of the composed communication system with respect to the environment. It is very unlikely that both parts send and receive a message at the same time instant (unless there is a special reason for forcing it), so the behaviour of the overall system is described by the parallel composition of the receiving and sending modules, together with the converter to be synthesized. Therefore, 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, the goal is to find a converter X whose composition with the sender and receiver A meets the system specification after hiding the internal signals: $A \diamond X \subseteq C$.

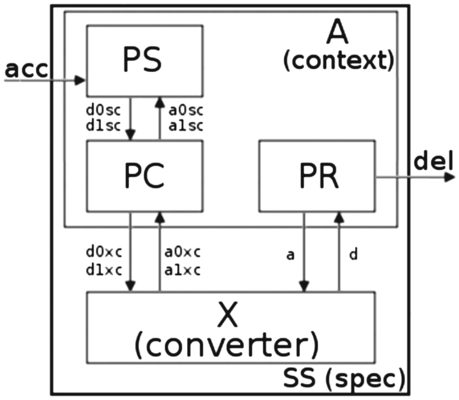

Fig. 2. Communication system described in Sect. 4.

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 was proposed originally in [22,37], and this exposition is adapted from [9]. 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 acknowledgment is also based on the alternating bit: each message received by either process in the system corresponds to an acknowledgment message that depends on the bit value. If the acknowledgment received by a process does not correspond to the message sent originally, the message is resent until the correct acknowledgment 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 acknowledgments. This means that neither messages nor their acknowledgments are distinguished by any flags as it happens with the alternating bit.

Figure 2 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 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 interfaces of the communication system with the environment (the users). 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, acknowledgment messages a delivered to the converter by the receiver are transformed into acknowledgments 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 finite automata which recognize prefix-closed regular languages, and solve the language equation $PS \diamond PC \diamond PR \diamond X \subseteq SS$, where $PS \diamond PC \diamond PR$ is the context, X is the unknown protocol converter and SS is the specification.

The largest automaton solution of the previous equation is obtained by the computation $S = \overline{PS \diamond PC \diamond PR \diamond \overline{SS}}$, that is the instantiation of the equation (3). The automaton solution can be obtained by running a few simple commands on BALM-II (like `complement, product` (i.e., intersection), `expansion` and `restriction`).

The automata of the components of the communication system and of the largest prefix-closed solution of the converter problem can be found in [10], with a comparison with respect to the other solutions reported in the literature.

The protocol conversion problem was addressed in [22], as an instance of supervisory control of discrete event systems, where the converter language is restricted to be a sublanguage of the context A, and in [37] with the formalism of input-output automata. In [22] the problem is modeled by the equation $A \diamond X = C$ over regular languages with the rectification topology. The solution is given as a sublanguage of A of the form $A \diamond C \setminus A \diamond \overline{C}$ (not the largest solution). An algorithm to obtain the largest compositionally progressive solution is provided that first splits the states of the automaton of the unrestricted solution (refining procedure, exponential step due to the restriction operator), and then deletes the states that violate the desired requirement of progressive composition (linear step). This algorithm does not generalize, as it is, to topologies where the unknown component depends also on signals that do not appear in the component A.

5 Conclusions

We transformed the protocol converter synthesis problem into solving inequations and equations over regular languages represented by finite automata and finite state machines. The solutions are computed with BALM-II, a software package that finds the largest solution of parallel and synchronous inequations and equations over automata and FSMs. For illustration we showed how to solve a classical problem to design a protocol converter for interfacing an alternating-bit sender and a non-sequenced receiver; we reported an example with parallel composition and one with synchronous composition. The ability to derive all protocol converters that solve the problem is an advantage over computational techniques that deliver only one or a few solutions (which might yield inferior implementations, e.g., as sequential circuits). For simplicity in this paper we dealt only with inequations, but we could filter out the solutions that do not satisfy the strict equality, thus solving strict equations and avoiding trivial empty solutions. Notice that our approach works as it is also for non-deterministic specifications, and in general delivers a non-deterministic result representing a collection of deterministic solutions. Our current algorithm deals only with FA and FSMs, where all data are represented explicitly. In order to deal with real protocols having data words (with 8, 16, 32, 64 bits) there must be some abstraction mechanism, which is out of scope for this paper, which is focussed on proposing an automatic converter synthesis procedure for the control part. We will investigate as future work how to handle data paths on top of our proposed approach for control logic, starting from the techniques proposed in the literature. Another related practical issue is scalability. The worst-case of the proposed algorithm is exponential in the input state space, however this worst-case in practice is achieved only in pathological examples. To assess the practicality of the proposed methods and compare them, the availability of an established collection of shared benchmarks would help the research community.

References

1. Sinha, R., Roop, P.S., Salcic, Z., Basu, S.: Correct-by-construction multi-component SoC design. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2012, San Jose, CA, USA. EDA Consortium, pp. 647–652 (2012)
2. Martin, G., Bailey, B., Piziali, A.: ESL Design and Verification: A Prescription for Electronic System Level Methodology. Morgan Kaufmann, San Mateo (2007)
3. Passerone, R., Rowson, J.A., Sangiovanni-Vincentelli, A.: Automatic synthesis of interfaces between incompatible protocols. In: Proceedings of the 35th Annual Design Automation Conference, DAC 1998. ACM, New York (1998). http://doi.acm.org/10.1145/277044.277047
4. Petrenko, A., Yevtushenko, N.: Solving asynchronous equations. In: Budkowski, S., Cavalli, A., Najm, E. (eds.) FORTE XI/PSTV XVIII 1998, pp. 231–247. Kluwer Academic Publishers, Dordrecht (1998)
5. Yevtushenko, N., Villa, T., Brayton, R., Petrenko, A., Sangiovanni-Vincentelli, A.: Solution of parallel language equations for logic synthesis. In: The Proceedings of the International Conference on Computer-Aided Design, pp. 103–110, November 2001
6. Yevtushenko, N., Villa, T., Brayton, R., Petrenko, A., Sangiovanni-Vincentelli, A.: Solution of synchronous language equations for logic synthesis. In: The Biannual 4th Russian Conference with Foreign Participation on Computer-Aided Technologies in Applied Mathematics, September 2002
7. Yevtushenko, N., Villa, T., Brayton, R., Petrenko, A., Sangiovanni-Vincentelli, A.: Sequential synthesis by language equation solving, Tech. Report No. UCB/ERL M03/9, Berkeley, CA, April 2003

8. Yevtushenko, N., Villa, T., Zharikova, S.: Solving language equations over synchronous and parallel composition operators. In: Kunc, M., Okhotin, A. (eds.) Proceedings of the 1st International Workshop on Theory and Applications of Language Equations, TALE 2007, Turku, Finland, 2 July 2007, pp. 14–32. Turku Centre for Computer Science (2007)
9. Villa, T., Yevtushenko, N., Brayton, R., Mishchenko, A., Petrenko, A., Sangiovanni-Vincentelli, A.: The Unknown Component Problem: Theory and Applications. Springer, New York (2012)
10. Castagnetti, G., Piccolo, M., Villa, T., Yevtushenko, N., Mishchenko, A., Brayton, R.K. : Solving parallel equations with BALM-II, EECS Department, University of California, Berkeley, Tech. Report UCB/EECS-2012-181, July 2012. http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-181.html
11. Castagnetti, G., Piccolo, M., Villa, T.: BALM-II. http://esd.scienze.univr.it/index.php/it/balm-ii.html
12. B. R. Group.: BALM, website and User's Manual. http://embedded.eecs.berkeley.edu/Respep/Research/mvsis/balm.html
13. Androutsopoulos, V., Brookes, D., Clarke, T.: Protocol converter synthesis. Comput. Digital Tech. IEE Proc. **151**(6), 391–401 (2004)
14. Jiang, Y., Jin, Y.: Protocol converter sysnthesis: an application of control synthesis, EE219C Class Project Report, December 1999
15. Passerone, R., Rowson, J.A., Sangiovanni-Vincentelli, A.L.: Automatic synthesis of interfaces between incompatible protocols. In: DAC, pp. 8–13 (1998)
16. Passerone, R., de Alfaro, L., Henzinger, T.A., Sangiovanni-Vincentelli, A.L.: Convertibility verification and converter synthesis: two faces of the same coin. In: ICCAD, pp. 132–139 (2002)
17. Passerone, R.: Semantic foundations for heterogeneous systems, Ph.D. dissertation, EECS Department, University of California, Berkeley, Tech. Report No. UCB/ERL M98/30 (2004). http://www.eecs.berkeley.edu/Pubs/TechRpts/1998/3445.html
18. Passerone, R.: Interface specification and converter synthesis. In: Zurawski, R. (ed.) Embedded Systems Handbook. CRC Press, Taylor and Francis Group, Boca Raton (2005)
19. Watanabe, S., Seto, K., Ishikawa, Y., Komatsu, S., Fujita, M.: Protocol transducer synthesis using divide and conquer approach. In: Design Automation Conference, 2007, ASP-DAC 2007, Asia, South Pacific, pp. 280–285, January 2007
20. Bhaduri, P., Ramesh, S.: Interface synthesis and protocol conversion. Formal Aspects Comput. **20**, 205–224 (2008)
21. Avnit, K., Sowmya, A.: A formal approach to design space exploration of protocol converters. In: The Proceedings of the Design, Automation and Test in Europe Conference, pp. 129–134, April 2009
22. Kumar, R., Nelvagal, S., Marcus, S.: A discrete event systems approach for protocol conversion. Discrete Event Dyn. Syst. Theory Appl. **7**(3), 295–315 (1997)
23. Sinha, R., Girault, A., Goessler, G., Roop, P.S.: A formal approach to incremental converter synthesis for system-on-chip design. ACM Trans. Des. Autom. Electron. Syst. **20**(1), 13:1–13:30 (2014). http://doi.acm.org/10.1145/2663344
24. Buffalov, S., El-Fakih, K., Yevtushenko, N., Bochmann, G.: Progressive solutions to a parallel automata equation. In: König, H., Heiner, M., Wolisz, A. (eds.) FORTE 2003. LNCS, vol. 2767. Springer, Heidelberg (2003)
25. El-Fakih, K., Buffalov, S., Yevtushenko, N., Bochmann, G.: Progressive solutions to a parallel automata equation. Theoret. Comput. Sci. **362**, 17–32 (2006)
26. DiBenedetto, M.D., Sangiovanni-Vincentelli, A., Villa, T.: Model matching for finite state machines. IEEE Trans. Autom. Control **46**(11), 1726–1743 (2001)

27. Cao, J., Nymeyer, A.: Formal model of a protocol converter. In: Downey, R., Manyem, P. (eds.) Fifteenth Computing: The Australasian Theory Symposium (CATS 2009), CRPIT, vol. 94. ACS, Wellington, pp. 107–117 (2009)
28. Sinha, R., Roop, P., Basu, S.: A model checking approach to protocol conversion. In: Workshop on Model-driven High-level Programming of Embedded Systems (2007)
29. Sinha, R., Roop, P.S., Basu, S.: SoC design approach using convertibility verification. EURASIP J. Emb. Sys. **2008** (2008)
30. Sinha, R., Roop, P.S., Basu, S., Salcic, Z.: Multi-clock SoC design using protocol conversion. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2009. European Design and Automation Association, pp. 123–128 (2009)
31. Sinha, R.: Automated techniques for formal verification of SoCs, Ph.D. dissertation, University of Auckland, New Zealand (2009)
32. Autili, M., Inverardi, P., Mignosi, F., Spalazzese, R., Tivoli, M.: Automated synthesis of application-layer connectors from automata-based specifications. In: Dediu, A.-H., Formenti, E., Martín-Vide, C., Truthe, B. (eds.) LATA 2015. LNCS, vol. 8977, pp. 3–24. Springer, Heidelberg (2015)
33. Ciancia, V., Martin, J., Martinelli, F., Matteucci, I., Petrocchi, M., Pimentel, E.: Automated synthesis and ranking of secure BPMN orchestrators. Int. J. Secur. Softw. Eng. **5**(2), 44–64 (2014). http://dx.doi.org/10.4018/ijsse.2014040103
34. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing Company, Reading (2001)
35. Lynch, N., Tuttle, M.: An introduction to input/output automata. CWI-Q. **2**(3), 219–246 (1989)
36. M. R. Group: MVSIS, software package. http://embedded.eecs.berkeley.edu/Respep/Research/mvsis/software.html
37. Hallal, H., Negulescu, R., Petrenko, A.: Design of divergence-free protocol converters using supervisory control techniques. In: 7th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2000, vol. 2, pp. 705–708, December 2000
38. Avnit, K., D'Silva, V., Sowmya, A., Ramesh, S., Parameswaran, S.: A formal approach to the protocol converter problem. In: DATE, pp. 294–299 (2008)
39. D'Silva, V., Ramesh, S., Sowmya, A.: Bridge over troubled wrappers: Automated interface synthesis. In: Proceedings of the 17th International Conference on VLSI Design, VLSID 2004. IEEE Computer Society, Washington, D.C., p. 189 (2004). http://dl.acm.org/citation.cfm?id=962758.963411
40. Tivoli, M., Fradet, P., Girault, A., Gößler, G.: Adaptor synthesis for real-time components. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 185–200. Springer, Heidelberg (2007)