Synthesis of Interface Automata

Purandar Bhaduri

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati 781039, India
Email: pbhaduri@iitg.ernet.in

Abstract. We investigate the problem of synthesising an interface automaton R such that $P \parallel R \preceq Q$, for given deterministic interface automata P and Q. We show that a solution exists iff P and $Q^{\perp}$ are compatible, and the most general solution is given by $(P \parallel Q^{\perp})^{\perp}$, where $P^{\perp}$ is the automaton P with inputs and outputs interchanged. We also characterise solutions in terms of winning input strategies in the automaton $(P \otimes Q^{\perp})^{\perp}$, and the most general solution in terms of the most permissive winning strategy. We apply the synthesis problem for interfaces to the problem of synthesising converters for mismatched protocols.

1 Introduction

Interfaces play a central role in component based design and verification of systems. In this paper we study the problem of synthesising an interface R, which composed with a known interface P is a refinement of an interface Q. This is a central problem in component based top-down design of a system. The interface Q is an abstract interface, a high level specification of the component under development. The interface P is a known part of the implementation and we are required to find the most general (i.e., abstract) solution R satisfying the relation $P \parallel R \preceq Q$. Here $P \parallel Q$ is the composition of P and Q, and $P \preceq Q$ denotes 'P is a refinement of Q'. This problem has wide ranging applications from logic synthesis to the design of discrete controllers, and has been studied previously in [20, 21], where the composition is either the synchronous or parallel composition of languages, and refinement is inclusion. We study the problem in the setting of interface automata [6], where composition and refinement of interfaces are respectively the composition of interface automata and alternating refinement relations[2].

Interface automata are like ordinary automata, except for the distinction between input and output actions. The input actions of an interface automaton P are controlled by its environment. Therefore an input action labelling a transition is an *input assumption* (or constraint on P's environment). Dually, an output action of P is under P's control, and represents an an *output guarantee* of P. Note that unlike I/O automata [12], interface automata are *not* required to be input enabled. If an input action a is not enabled at a state s, it is an assumption

on the automaton's environment that it will not provide a as an input in state s.

When two interfaces P and Q are composed, the combined interface may contain incompatible states: states where one interface can generate an output that is not a legal input for the other. In the combined interface it is the environment's responsibility to ensure that such a state is unreachable [6]. This can be formalised as a two person game [6] which has the same flavour as the controller synthesis problem of Ramadge and Wonham [17]; in our setting the role of the controller is played by the environment. More formally, we follow de Alfaro [7] in modelling an interface as a game between two players, Output and Input. Player Output represents the system and its moves represent the outputs generated by the system. Player Input represents the environment; its moves represent the inputs the system receives from its environment. In general, the set of available moves of each player depends on the current state of the combined system. The interface is well-formed if the Input player has a winning strategy in the game, where the winning condition is to avoid all incompatible states. Clearly, the game aspect is relevant only when defining the composition of two interfaces.

Refinement of interfaces corresponds to weakening assumptions and strengthening guarantees. An interface P refines Q only if P can be used in any environment where Q can be. The usual notion of refinement is *simulation* or *trace containment* [12]. For interface automata, a more appropriate notion is that of *alternating simulation* [2], which is contravariant on inputs and covariant on outputs: if $P \preceq Q$ (P *refines* Q), P accepts more inputs (weaker input assumptions) and provides fewer outputs (stronger output guarantees). Thus alternating refinement preserves compatibility: if P and Q are compatible (i.e., $P \parallel Q$ is well-formed) and $P' \preceq P$, then so are P' and Q.

In this paper we show that a solution to $P \parallel R \preceq Q$ for R exists for deterministic interface automata iff P and $Q^\perp$ are compatible, and the most abstract (under alternating refinement) solution is given by $(P \parallel Q^\perp)^\perp$. Further, such an R can be constructed from the most permissive winning strategy for player Input in the combined game $(P \otimes Q^\perp)^\perp$. Here $P^\perp$ is the game P with the moves of the players Input and and Output interchanged, and $P \otimes Q$ is the combined game obtained from P and Q by synchronising on shared actions and interleaving the rest. We say a strategy π is more permissive than π' when, at every position in the game, the set of moves allowed by π includes those allowed by π'. The most permissive winning strategy is one that is least restrictive. This result ties up the relation between composition, refinement, synthesis and winning strategies, and should be seen as one more step towards a "uniform framework for the study of control, verification, component-based design, and implementation of open systems", based on games [7].

Note that the notation $P^\perp$ is borrowed from linear logic [8], where games play an important semantic role [3]. Using the notation of linear logic, the solution R to the synthesis problem can be written as $(P \otimes Q^\perp)^\perp = P^\perp \parr Q = P \multimap Q$, where $\otimes$, $\parr$ and $\multimap$ are respectively, the linear logic connectives 'With', 'Par' and linear

implication. In our setting, the $\otimes$ connective of linear logic is parallel composition $\|$. The striking similarity of this solution with the language equation posed in [20, 21] is intriguing. In their framework, the largest solution of the language equation $P \bullet R \subseteq Q$ for R is the language $\overline{P \bullet \overline{Q}}$ where $P \bullet Q$ is the synchronous (or parallel) composition of languages P and Q, and $\overline{P}$ is the complement of P. Clearly, there is a formal correspondence between $P \bullet Q$ and our $P \parallel Q$, between $\overline{P}$ and our $P^{\perp}$, and between language inclusion and alternating simulation.

We should also mention the formal resemblance of our work with Abramsky's Semantics of Interaction [1], based on the game semantics of linear logic. In particular, the strategy called *Application* (or *Modus Ponens*) in [1] is the solution to our synthesis problem in a different setting. The solution $R = P \multimap Q$ suggests that the problem of synthesis can be seen as the construction of a suitable morphism in an appropriate category of interface automata, along the lines of [13, 18]. However, we do not pursue this thread in this paper.

As a practical application we show how to apply interface synthesis to the protocol conversion problem for mismatched network protocols. The heterogeneity of existing networks often results in incompatible protocols trying to communicate with each other. The protocol conversion problem is, given two network protocols P_1 and P_2 which are mismatched, to come up with a converter C which mediates between the two protocols, such that the combined system conforms to an overall specification S. We show that a converter C, if it exists, can be obtained as the solution to $P \parallel C \preceq S$, where $P = P_1 \parallel P_2$ is the composition of the two protocols.

The controller synthesis problem and its solution as a winning strategy in a game has a long history, going back to Büchi and Landwebers' solution of Church's problem [4]. More recent applications of the idea in the synthesis of open systems occur in [13, 14, 16]. The control of discrete event systems [17] and the synthesis of converters for mismatched protocols [15] can be seen as applications of the same general principle. The present work extends the principle to the composition and refinement of interfaces.

2 Interface Automata

In this section we define interface automata and their composition and refinement. We follow the game formulation presented in [7]. Throughout this work we consider only *deterministic* interface automata.

Definition 1. *An* interface automaton P *is a tuple* $(S_P, S_P^0, \mathcal{A}_P^I, \mathcal{A}_P^O, \Gamma_P^I, \Gamma_P^O, \delta_P)$ *where:*

- S_P *is a finite set of* states.
- $S_P^0 \subseteq S_P$ *is the set of* initial states, *which has at most one element, denoted* s_P^0.
- $\mathcal{A}_P^I$ *and* $\mathcal{A}_P^O$ *are disjoint sets of* input *and* output actions. *The set* $\mathcal{A}_P = \mathcal{A}_P^I \cup \mathcal{A}_P^O$ *is the set of all* actions.

– $\Gamma_P^I : S_P \to 2^{\mathcal{A}_P^I}$ *is a map assigning to each state* $s \in S_P$ *a set (possibly empty) of* input moves. *Similarly,* $\Gamma_P^O : S_P \to 2^{\mathcal{A}_P^O}$ *assigns to each state* $s \in S_P$ *a set (again, possibly empty) of* output moves. *The input and output moves at a state* s *correspond to actions that can be accepted and generated at* s *respectively. Denote by* $\Gamma_P(s) = \Gamma_P^I(s) \cup \Gamma_P^O(s)$ *the set of all actions at* s.
– $\delta_P : S_P \times \mathcal{A}_P \to S_P$ *is a* transition function *associating a target state* $\delta_P(s, a)$ *with each state* $s \in S_P$ *and action* $a \in \mathcal{A}_P$. *Note that the value* $\delta_P(s, a)$ *makes sense only when* $a \in \Gamma_P(s)$. *When* $a \notin \Gamma_P(s)$, *the value can be arbitrary.*

The interface automaton P is said to be *empty* when its set of initial states S_P^0 is empty. Empty interface automata arise when incompatible automata are composed.

Definition 2. *An* input strategy *for* P *is a map* $\pi^I : S_P^+ \to 2^{\mathcal{A}_P^I}$ *satisfying* $\pi^I(\sigma s) \subseteq \Gamma_P^I(s)$ *for all* $s \in S_P$ *and* $\sigma \in S_P^*$. *An* output strategy $\pi^O : S_P^+ \to 2^{\mathcal{A}_P^O}$ *is defined similarly. The set of input and output strategies of* P *are denoted by* Π_P^I *and* Π_P^O *respectively.*

An input and output strategy jointly determine a set of traces in S_P^+ as follows. At each step, if the input strategy proposes a set $\mathcal{B}^I$ of actions, and the output strategy proposes a set $\mathcal{B}^O$ of actions, an action from $\mathcal{B}^I \cup \mathcal{B}^O$ is chosen nondeterministically.

Definition 3. *Given a state* $s \in S_P$, *and input strategy* π^I *and an output strategy* π^O, *the set* $Outcomes(s, \pi^I, \pi^O) \subseteq S_P^+$ *of resulting plays is defined inductively as follows:*

– $s \in Outcomes_P(s, \pi^I, \pi^O)$;
– *if* $\sigma t \in Outcomes(s, \pi^I, \pi^O)$ *for* $\sigma \in S_P^+$ *and* $t \in S_P$, *then for all* $a \in \pi^I(\sigma t) \cup \pi^O(\sigma t)$ *the sequence* $\sigma t \delta_P(s, a) \in Outcomes_P(s, \pi^I, \pi^O)$.

A state $s \in S_P$ is said to be *reachable* in P, if there is a sequence of states $s_0, s_1, \ldots, s_n$ with $s_0 \in S_P^0$, $s_n = s$, and for all $0 \leq k < n$ there is $a_k \in \Gamma_P(s_k)$ such that $\delta_P(s_k, a_k) = s_{k+1}$. Reach($P$) denotes the set of reachable states of P

The refinement of interface automata is known as *alternating simulation*, the right notion of simulation between games [2]. Intuitively, an alternating simulation $\rho \subseteq S_P \times S_Q$ from P to Q is a relation for which $(s, t) \in \rho$ implies all input moves from t can be simulated by s and all output moves from s can be simulated by t.

Definition 4. *An* alternating simulation ρ *from* P *to* Q *is a relation* $\rho \subseteq S_P \times S_Q$ *such that, for all* $(s, t) \in \rho$ *and all* $a \in \Gamma_Q^I(t) \cup \Gamma_P^O(s)$, *the following conditions are satisfied:*

1. $\Gamma_Q^I(t) \subseteq \Gamma_P^I(s)$;
2. $\Gamma_P^O(s) \subseteq \Gamma_Q^O(t)$;
3. $(\delta_P(s, a), \delta_Q(t, a)) \in \rho$.

Refinement between interface automata is defined as the existence of an alternating simulation between the initial states.

Definition 5. *An interface automaton P* refines *an interface automaton Q, written $P \preceq Q$, if the following conditions are satisfied:*

1. $\mathcal{A}_Q^I \subseteq \mathcal{A}_P^I$;
2. $\mathcal{A}_P^O \subseteq \mathcal{A}_Q^O$;
3. *there is an alternating simulation ρ from P to Q, such that $(s^0, t^0) \in \rho$ for some $s^0 \in S_P^0$ and $t^0 \in S_Q^0$.*

We now define the parallel composition $P \parallel Q$ of interface automata P and Q in a series of steps.

Definition 6. *P and Q are* composable *if $\mathcal{A}_P^O \cap \mathcal{A}_Q^O = \emptyset$.*

We first define the *product automaton* $P \otimes Q$ of two composable interface automata P and Q, by synchronising their shared actions and interleaving all others. The set of shared actions of P and Q is defined by $\text{Shared}(P, Q) = \mathcal{A}_P \cap \mathcal{A}_Q$.

Definition 7. *The* product *$P \otimes Q$ of two composable interface automata P and Q is defined by*

- $S_{P\otimes Q} = S_P \times S_Q$;
- $S_{P\otimes Q}^0 = S_P^0 \times S_Q^0$;
- $\mathcal{A}_{P\otimes Q}^I = (\mathcal{A}_P^I \cup \mathcal{A}_Q^I) \backslash \mathit{Comm}(P, Q)$ *where* $\mathit{Comm}(P, Q) = (\mathcal{A}_P^O \cap \mathcal{A}_Q^I) \cup (\mathcal{A}_P^I \cap \mathcal{A}_Q^O)$ *is the set of* communication actions*, a subset of* $\mathit{Shared}(P, Q)$;
- $\mathcal{A}_{P\otimes Q}^O = \mathcal{A}_P^O \cup \mathcal{A}_Q^O$;
- $\Gamma_{P\otimes Q}^I((s,t)) = (\Gamma_P^I(s) \backslash (\mathcal{A}_Q^O \cup \mathcal{A}_Q^I)) \cup (\Gamma_Q^I(t) \backslash (\mathcal{A}_P^O \cup \mathcal{A}_P^I)) \cup (\Gamma_P^I(s) \cap \Gamma_Q^I(t))$ *for all* $(s,t) \in S_P \times S_Q$;
- $\Gamma_{P\otimes Q}^O((s,t)) = \Gamma_P^O(s) \cup \Gamma_Q^O(t)$, *for all* $(s,t) \in S_P \times S_Q$;
- *for all* $a \in \mathcal{A}_{P\otimes Q}$,

$$\delta_{P\otimes Q}((s,t),a) = \begin{cases} (\delta_P(s,a), \delta_Q(t,a)) & \text{if } a \in \mathcal{A}_P \cap \mathcal{A}_Q \\ (\delta_P(s,a), t) & \text{if } a \in \mathcal{A}_P \backslash \mathcal{A}_Q \\ (s, \delta_Q(t,a)) & \text{if } a \in \mathcal{A}_Q \backslash \mathcal{A}_P \end{cases}$$

Since interface automata need not be input enabled, there may be reachable states in $P \otimes Q$ where a communication action can be output by one of the automaton but cannot be accepted as input by the other. These states are called *locally incompatible*.

Definition 8. *The set* $\mathit{Incomp}(P, Q)$ *of* locally incompatible states *of P and Q consists of all pairs $(s,t) \in S_P \times S_Q$ for which one of the following two conditions hold:*

1. $\exists a \in \mathit{Comm}(P, Q)$ *such that* $a \in \Gamma_P^O(s)$ *but* $a \notin \Gamma_Q^I(t)$,

2. $\exists a \in Comm(P,Q)$ *such that* $a \in \Gamma^O_Q(t)$ *but* $a \notin \Gamma^I_P(s)$.

A local incompatibility can be avoided if there is a helpful environment, which by providing the right sequence of inputs can steer the automaton away from such an undesirable state. The states from which Input can prevent the product $P \otimes Q$ from reaching a state in $\text{Incomp}(P,Q)$ are called *compatible*. In other words, the compatible states are those from which Input has a winning strategy. The calculation of winning strategy in such safety games, if one exists, by using the *controllable predecessors* of a set of states U and iterative refinement is standard [19].

Definition 9. *A state* $s \in S_{P\otimes Q}$ *is* compatible *if there is an input strategy* $\pi^I \in \Pi^I_{P\otimes Q}$ *such that, for all output strategies* $\pi^O \in \Pi^O_{P\otimes Q}$, *all* $\sigma \in Outcomes_{P\otimes Q}(s, \pi^I, \pi^O)$ *and all incompatible states* $w \in Incomp(P,Q)$, *the state* w *does not appear in the sequence* σ.

The composition $P \parallel Q$ is obtained by restricting $P \otimes Q$ to the states that can be reached from the initial state under an input strategy that avoids all locally incompatible states. We call these states *backward compatible*. These are the states that are reachable from the initial state of $P \otimes Q$ by visiting only compatible states. Note that in [7] backward compatible states are called *usably reachable states*.

Definition 10. *A state* $s \in S_{P\otimes Q}$ *is* backward compatible *in* $P \otimes Q$ *if there is an input strategy* $\pi^I \in \Pi^I_{P\otimes Q}$ *such that:*

- *for all initial states* $s_0 \in S^0_{P\otimes Q}$, *all output strategies* $\pi^O \in \Pi^O_{P\otimes Q}$, *all outcomes* $\sigma \in Outcomes_{P\otimes Q}(s_0, \pi^I, \pi^O)$ *and all* $w \in Incomp(P,Q)$, w *does not occur in* σ;
- *there is an initial state* $s_0 \in S^0_{P\otimes Q}$, *an output strategy* $\pi^O \in \Pi^O_{P\otimes Q}$, *and an outcome* $\sigma \in Outcomes_{P\otimes Q}(s_0, \pi^I, \pi^O)$ *such that* $s \in \sigma$.

Definition 11. *The* composition $P \parallel Q$ *of two interface automata* P *and* Q, *with* T *the set of backward compatible states of the product* $P \otimes Q$, *is an interface automaton defined by:*

- $S_{P\parallel Q} = T$
- $S^0_{P\parallel Q} = S^0_{P\otimes Q} \cap T$
- $\mathcal{A}^I_{P\parallel Q} = \mathcal{A}^I_{P\otimes Q}$
- $\mathcal{A}^O_{P\parallel Q} = \mathcal{A}^O_{P\otimes Q}$
- $\Gamma^I_{P\parallel Q}(s) = \{a \in \Gamma^I_{P\otimes Q}(s) \mid \delta_{P\otimes Q}(s,a) \in T\}$ *for all* $s \in T$
- $\Gamma^O_{P\parallel Q}(s) = \Gamma^O_{P\otimes Q}(s)$ *for all* $s \in T$
- *for all* $s \in T$, $a \in \Gamma_{P\parallel Q}(s)$,

$$\delta_{P\parallel Q}(s,a) = \begin{cases} \delta_{P\otimes Q}(s,a) \text{ if } \delta_{P\otimes Q}(s,a) \in T \\ \text{arbitrary} \quad \text{otherwise} \end{cases}$$

Definition 12. P *and* Q *are said to be* compatible *if their composition is non-empty i.e.,* $s^0_{P\parallel Q} \neq \emptyset$. *This is equivalent to* $s^0_{P\otimes Q} \in T$, *where* T *is the set of backward compatible states of* $P \otimes Q$.

Notation We write $\text{Reach}^O(P)$ to denote the set of states of P that are reachable from the initial state s^0_P by following only output actions.

We use the following lemma in our proof of Theorems 1 and 2 in Section 3. Since the best input strategy to avoid locally incompatible states is simply to generate no inputs to $P \otimes Q$ at any state, the set of compatible states in $P \otimes Q$ is simply the set of states from which $P \otimes Q$ cannot reach a state in $\text{Incomp}(P, Q)$ by a sequence of output actions.

Lemma 1. *P and Q are compatible iff the states in $\text{Reach}^O(P \otimes Q)$ are locally compatible, i.e., $\text{Reach}^O(P \otimes Q) \cap \text{Incomp}(P, Q) = \emptyset$.*

Proof. Suppose P and Q are compatible. Then $s^0_{P \otimes Q}$ is a backward compatible state in $P \otimes Q$. This implies there is an input strategy π^I for $P \otimes Q$ which avoids all locally incompatible states starting from $s^0_{P \otimes Q}$, no matter what the output strategy is. Now Output can always force $P \otimes Q$ to enter any state in $\text{Reach}^O(P \otimes Q)$. In other words, an output strategy π^O exists for which every state s in $\text{Reach}^O(P \otimes Q)$ appears in some sequence in $\text{Outcomes}_{P \otimes Q}(s^0_{P \otimes Q}, \pi^I, \pi^O)$. Since $s^0_{P \otimes Q}$ is a backward compatible in $P \otimes Q$, it follows that $\text{Reach}^O(P \otimes Q) \cap \text{Incomp}(P, Q) = \emptyset$. Conversely, suppose the states in $\text{Reach}^O(P \otimes Q)$ are locally compatible. This implies that any state in $\text{Incomp}(P, Q)$ can be reached, if at all, by following a sequence of actions which includes at least one input action. Then the input strategy which disables all such input actions avoids all locally incompatible states and so $s^0_{P \otimes Q}$ is backward compatible. □

3 Synthesis of Interface Automata

Our goal is to find the most general solution R to $P \parallel R \preceq Q$ when it exists, and characterise the conditions under which it exists. By a most general solution we mean, a solution U, such that for any solution V, it is the case that $V \preceq U$. In this section we prove our main result, viz., the most general solution to $P \parallel R \preceq Q$ is give by $R = (P \parallel Q^\perp)^\perp$ and a solution exists iff P and $Q^\perp$ are compatible. Here $P^\perp$ is the same as P, except all the input actions in P become output actions in $P^\perp$ and similarly the output actions of P are the input actions of $P^\perp$.

Example 1. Figure 1 presents three examples to illustrate the synthesis idea with given interface automata P and Q. The construction of $Q^\perp$, $P \parallel Q^\perp$ and $R = (P \parallel Q^\perp)^\perp$ are shown in each case.

1. In Figure 1(a), the input actions are $\mathcal{A}^I_P = \mathcal{A}^I_Q = \{a, c\}$, and the output actions are $\mathcal{A}^O_P = \mathcal{A}^O_Q = \{b, d\}$. Note that in $P \parallel Q^\perp$, the transition labelled $c?$ does not appear, as it is a shared action, and has to be present in both P and $Q^\perp$ to appear in their product. Note also, how b appears as an input action in the result $(P \parallel Q^\perp)^\perp$.
2. In Figure 1(b), the action sets are $\mathcal{A}^I_P = \{a\}$, $\mathcal{A}^O_P = \{b\}$, $\mathcal{A}^I_Q = \{a, c\}$ and $\mathcal{A}^O_Q = \{b, d\}$. In this case, the solution is essentially identical with Q,

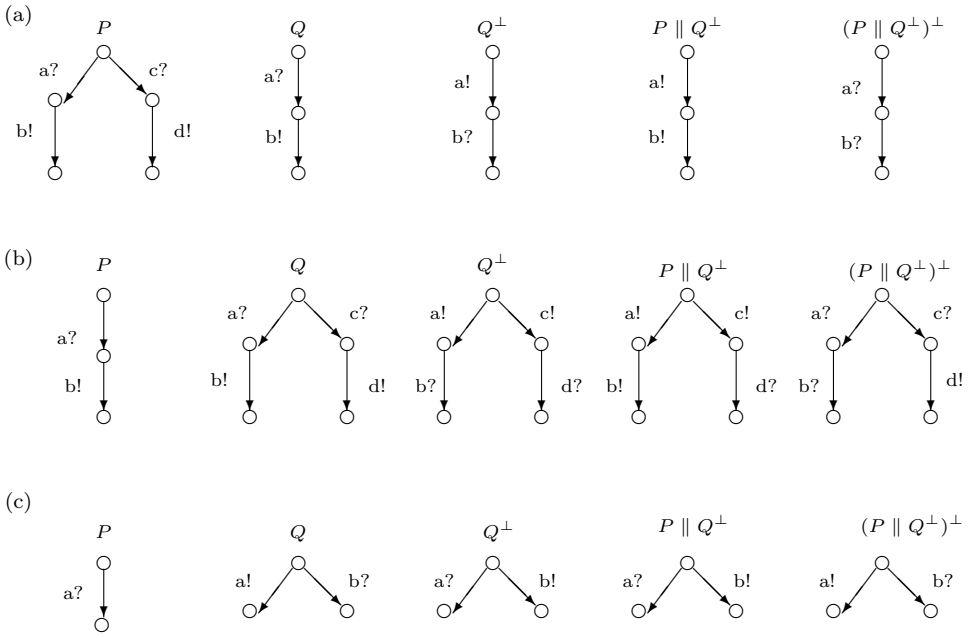


Fig. 1. Interface Automata Synthesis Examples

except for the polarity of action b. Note that there is already an alternating simulation between P and Q. The input transition labelled $b?$ appears in R because we assume $\mathcal{A}_P^O \subseteq \mathcal{A}_R^I$: in some sense, R can be thought of as a controller for P, and hence should be allowed to use all the output actions of P as input, in addition to driving the input actions of P. Note that if we changed the the input action set of P to be $\mathcal{A}_P^I = \mathcal{A}_Q^I = \{a, c\}$, then there would be no solution R, because P and $Q^\perp$ would not be compatible: in the initial state, $Q^\perp$ is ready to output a c, but P is not ready to accept it as input, even though c is a communication action between the two.

3. In Figure 1(c), the action sets are $\mathcal{A}_P^I = \{a\}$, $\mathcal{A}_P^O = \emptyset$, $\mathcal{A}_Q^I = \{b\}$ and $\mathcal{A}_Q^O = \{a\}$. In this example, an input of P appears as an output of Q. The result $(P \parallel Q^\perp)^\perp$ adds the input b and also converts a from an input to an output. In this case, R is identical to Q.

Note Throughout this section we make the weak assumption that $\mathcal{A}_P^I \subseteq \mathcal{A}_Q^I \cup \mathcal{A}_Q^O$. This is to ensure that an environment E for which $Q \parallel E$ is a closed system (i.e., has no inputs) will also make $(P \parallel R) \parallel E$ a closed system. So any inputs to P will be provided by an output from the environment of Q or from R. In the latter case, such an input of P will be an output of Q. Further, we assume that the solution R satisfies $\mathcal{A}_P^O \subseteq \mathcal{A}_R^I$. This is to allow R to use the output actions of P as inputs in carrying out its control objectives. It is clear that any solution

R will satisfy $\mathcal{A}_R^O \subseteq \mathcal{A}_Q^O \backslash \mathcal{A}_P^O$, and for the most general solution the two sets will be equal.

Notation We write $p \xrightarrow{a} p'$ if $a \in \Gamma_P(p)$ and $\delta(p,a) = p'$ for states p, p' and action a in an interface automaton P. We call $p \xrightarrow{a} p'$ an *input transition* if a is an input action of P. An output transition is defined similarly.

First we prove a result about compatibility that is used in Theorem 1 below.

Lemma 2. *If P and $Q^\perp$ are compatible, then P and $(P \parallel Q^\perp)^\perp$ are compatible.*

Proof. Suppose P and $Q^\perp$ are compatible, but P and $(P \parallel Q^\perp)^\perp$ are not. By Lemma 1, this means there exists a state $(p,(p',q)) \in \text{Reach}^O(P \otimes (P \parallel Q^\perp)^\perp)$ which is in $\text{Incomp}(P,(P \parallel Q^\perp)^\perp)$. Since the interface automata we consider are deterministic, it must be the case that $p = p'$. This implies that there exists an $a \in \text{Comm}(P,(P \parallel Q^\perp)^\perp)$ such that either (a) $a \in \Gamma_P^O(p)$ and $a \notin \Gamma^I_{(P\parallel Q^\perp)^\perp}(p,q) = \Gamma^O_{(P\parallel Q^\perp)}(p,q) = \Gamma_P^O(p) \cup \Gamma_Q^I(q)$, which is impossible, or (b) $a \notin \Gamma_P^I(p)$ and $a \in \Gamma^O_{(P\parallel Q^\perp)^\perp}(p,q)$ which implies $(p,q) \xrightarrow{a} (p',q')$ is an input transition in $P \parallel Q^\perp$ and $p \xrightarrow{a} p'$ is not an input transition in P. This is possible only if $a \in \mathcal{A}_Q^O$ but $a \notin \mathcal{A}_P^I$, which contradicts our assumption that $a \in \text{Comm}(P,(P \parallel Q^\perp)^\perp)$. □

Theorem 1. *A solution R to $P \parallel R \preceq Q$ exists iff P and $Q^\perp$ are compatible.*

Proof. (If) Suppose P and $Q^\perp$ are compatible. By Lemma 2 so are P and $(P \parallel Q^\perp)^\perp$. Take $R = (P \parallel Q^\perp)^\perp$. We show that there exists an alternating simulation ρ between $P \parallel R$ and Q. Define the relation $\rho = \{((p,(p,q)),q) \mid (p,(p,q))$ is a state in $P \parallel R\}$. Since (s_P^0, s_Q^0) is the initial state of R, $(s_P^0,(s_P^0,s_Q^0))$ is the initial state of $P \parallel R$, and hence $((s_P^0,(s_P^0,s_Q^0)),s_Q^0)$ is in ρ. Now suppose $((p,(p,q)),q) \in \rho$ and $q \xrightarrow{a} q$ is an input transition in Q. It follows that $q \xrightarrow{a} q'$ is an output transition in $Q^\perp$. Therefore, $p \xrightarrow{a} p'$ is an input transition in P for some p', since (p,q), being in $P \parallel Q^\perp$, is backward compatible in $P \otimes Q^\perp$. Hence $(p,q) \xrightarrow{a} (p',q')$ is an output transition in $P \parallel Q^\perp$, and so an input transition in $(P \parallel Q^\perp)^\perp$, whence $(p,(p,q)) \xrightarrow{a} (p',(p',q'))$ is an input transition in $P \parallel (P \parallel Q^\perp)^\perp$ and by definition of ρ, $((p',(p',q')),q')$ is again in ρ. Similarly for the output side, suppose $((p,(p,q)),q) \in \rho$ and $(p,(p,q)) \xrightarrow{a} (p',(p'',q'))$ is an output transition in $P \parallel (P \parallel Q^\perp)^\perp$. Since we consider only deterministic automata, $p' = p''$. Also, it must be the case that $a \in \text{Comm}(P,(P \parallel Q^\perp)^\perp)$, because an output action of P is an output action of $P \parallel Q^\perp$, and therefore an input action of $(P \parallel Q^\perp)^\perp$. Suppose $p \xrightarrow{a} p'$ is an output transition in P, and because P and $Q^\perp$ are compatible, and (p,q) is backward compatible in $P \otimes Q^\perp$, $q \xrightarrow{a} q'$ is an input transition in $Q^\perp$, and hence an output transition in Q. On the other hand, if $p \xrightarrow{a} p'$ is an input transition in P, then since $(p,(p,q)) \xrightarrow{a} (p',(p',q'))$ is an output transition in $P \parallel (P \parallel Q^\perp)^\perp$, $(p,q) \xrightarrow{a} (p',q')$ is an output transition in $(P \parallel Q^\perp)^\perp$, and therefore an input transition in $(P \parallel Q^\perp)$. From the assumption that $\mathcal{A}_P^I \subseteq \mathcal{A}_Q^I$ and by the definition of the product $P \otimes Q^\perp$ it follows

that $q \xrightarrow{a} q'$ is an input transition of $Q^{\perp}$, and hence an output transition of Q. By the definition of ρ, $((p', (p', q')), q') \in \rho$, hence ρ is an alternating simulation as required.

(Only if) We show the contrapositive. Suppose P and $Q^{\perp}$ are not compatible. Then, by Lemma 1, there exists a state $(p, q) \in \text{Reach}^{O}(P, Q)$ which is incompatible, i.e., there is an a such that either (a) $a \in \Gamma^{O}_{P}(p)$ and $a \notin \Gamma^{O}_{Q}(q)$ or (b) $a \notin \Gamma^{I}_{P}(p)$ and $a \in \Gamma^{I}_{Q}(q)$. Both possibilities rule out the existence of an alternating simulation between $P \parallel R$ and Q for any R.

□

Theorem 2.
When the condition stated in Theorem 1 is satisfied, the most general solution to $P \parallel R \preceq Q$ is $R = (P \parallel Q^{\perp})^{\perp}$.

Proof. In the proof of Theorem 1 (If part) we have already shown that $R = (P \parallel Q^{\perp})^{\perp}$ is a solution. Suppose U is any solution to $P \parallel R \preceq Q$. We construct an alternating simulation ν from U to $(P \parallel Q^{\perp})^{\perp}$ as follows. By assumption, there exists an alternating simulation ρ from $P \parallel U$ and Q. Define $\nu = \{(u, (p, q)) \mid ((p, u), q) \in \rho\}$. Clearly $(s^0_U, (s^0_P, s^0_Q)) \in \nu$, since $((s^0_P, s^0_U), s^0_Q) \in \rho$. Now suppose $(u, (p, q)) \in \nu$ and $u \xrightarrow{a} u'$ is an output transition in U. This implies $p \xrightarrow{a} p'$ is an input transition in P for some p', since by assumption $((p, u), q) \in \rho$ and therefore (p, u) is backward compatible in $P \otimes U$. Hence, $(p, u) \xrightarrow{a} (p', u')$ is an output transition in $P \parallel U$. It follows that $q \xrightarrow{a} q'$ is an output transition in Q for some q', with $((p', u'), q') \in \rho$, which is equivalent to $q \xrightarrow{a} q'$ is an input transition in $Q^{\perp}$. Therefore, $(p, q) \xrightarrow{a} (p', q')$ is an input transition in $P \parallel Q^{\perp}$, since (p, q) is backward compatible in $P \otimes Q^{\perp}$ by assumption. It follows that $(p, q) \xrightarrow{a} (p', q')$ is an output transition in $(P \parallel Q^{\perp})^{\perp}$ and $(u', (p', q')) \in \nu$ as required. Next suppose $(u, (p, q)) \in \nu$ and $(p, q) \xrightarrow{a} (p', q')$ is an input transition in $(P \parallel Q^{\perp})^{\perp}$, which is the same as $(p, q) \xrightarrow{a} (p', q')$ is an output transition in $P \parallel Q^{\perp}$. This implies that either (a) $p \xrightarrow{a} p'$ is an input transition in P and $q \xrightarrow{a} q'$ is an input transition in Q or (b) $p \xrightarrow{a} p'$ is an output transition in P and $q \xrightarrow{a} q'$ is an output transition in Q. For the first case, by the existence of the alternating simulation ρ, $(p, u) \xrightarrow{a} (p', u')$ is an input transition in $P \parallel U$ for some state u' in U with $((p', u'), q') \in \rho$ and hence $(u', (p', q')) \in \nu$. For the second case, $u \xrightarrow{a} u'$ is an input transition in U for some u', since (p, u) is backward compatible in $P \otimes U$. Further $(u', (p', q')) \in \nu$, since $((p', u'), q') \in \rho$, and the conclusion follows. □

4 Winning Strategies and Synthesis

We now characterise the most general solution to $P \parallel R \preceq Q$ in terms of winning strategies. Specifically, we show that the most general solution corresponds to the *most permissive* winning strategy for Input in $P \otimes (P \parallel Q^{\perp})^{\perp}$.

First we define winning strategies for Input and Output in games corresponding to the product $P \otimes Q$ of two interface automata P and Q. We also define a

natural partial order $\sqsubseteq^I$ on input strategies, such that $\sigma^I_P \sqsubseteq^I \tau^I_P$ if the strategy τ^I_P generates more inputs than σ^I_P at every state of P. A similar order $\sqsubseteq^O$ is defined on output strategies. Since the orders are lattices, the *most permissive strategy* exists, as is given by the lattice join. We then show that the parallel composition $P \parallel Q$ can be extracted from the most permissive winning strategy for Input.

Definition 13. *Let P and Q be composable interface automata. A* winning input strategy *for $P \otimes Q$ is an input strategy π^I such that for all output strategies π^O, all initial states $s_0 \in S^0_{P\otimes Q}$, all $\sigma \in Outcomes_{P\otimes Q}(s_0, \pi^I, \pi^O)$, and all incompatible states $w \in Incomp(P, Q)$, the state w does not appear in the sequence σ. The definition of a winning output strategy is symmetric, where the winning condition is that a state in Incomp(P,Q) must be reached in every run $\sigma \in Outcomes_{P\otimes Q}(s_0, \pi^I, \pi^O)$.*

We now define the order $\sqsubseteq$ on strategies. The idea is that an input strategy is higher in the order if it accepts more inputs. Dually an output strategy is higher in the order if it generates more outputs.

Definition 14. *The binary relation $\sqsubseteq^I$ on input strategies for P is defined by $\pi^I_0 \sqsubseteq \pi^I_1$ iff $\pi^I_0(\sigma) \subseteq \pi^I_1(\sigma)$ for all $\sigma \in S^+_P$. When $\pi^I_0 \sqsubseteq \pi^I_1$, we say π^I_1 is* more permissive than *π^I_0. Similarly, for output strategies, $\pi^O_0 \sqsubseteq^O \pi^O_1$ iff $\pi^O_0(\sigma) \subseteq \pi^O_1(\sigma)$ for all $\sigma \in S^+_P$.*

Clearly, the relations $\sqsubseteq^I$ and $\sqsubseteq^O$ are lattices, with top elements $\pi^I_T(\sigma s) = \Gamma^I_P(s)$ and $\pi^O_T(\sigma s) = \Gamma^O_P(s)$, and join and meet given by pointwise union and intersection. Note that the bottom elements are the empty strategies, which are allowed by the definition of strategies.

Corollary 1. *If there is a winning strategy for either player in a game then there is a most permissive winning strategy for that player.*

Proof. Simply take the join of the set of all winning strategies for the player. □

Next we show how to extract an interface automaton $\pi^I(P \otimes Q)$ from an input strategy π^I for the game $P \otimes Q$, by cutting down some of its states and transitions.

Definition 15. *The interface automaton $\pi^I(P \otimes Q)$* defined by input strategy *π^I for the game $P \otimes Q$ is defined as follows. Its set of input and output actions are the same as those of $P \otimes Q$. The set $S_{\pi^I(P\otimes Q)}$ contains those states of $P \otimes Q$ that are reached in some sequence in $Outcomes_{P\otimes Q}(s^0_{P\otimes Q}, \pi^I, \pi^O_T)$, where π^O_T is the top output strategy in the lattice of strategies (the one that produces the most output). The input moves of $\pi^I(P \otimes Q)$ are defined by $\Gamma^I(s) = \{a \mid a \in \Gamma^I_{P\otimes Q}$ such that $a \in \pi^I(\sigma s)$ for some $\sigma \in S^+_{\pi^I(P\otimes Q)}\}$. The input transitions of $\pi^I(P \otimes Q)$ are defined by $\delta(s, a) = \delta_{P\otimes Q}(s, a)$ when $a \in \Gamma^I(s)$ and an arbitrary element of $S_{\pi^I(P\otimes Q)}$ otherwise. The output moves and transitions are the straightforward restrictions of the output moves and transitions of $P \otimes Q$ to the set of states $S_{\pi^I(P\otimes Q)}$.*

The following proposition states that the parallel composition $P \parallel Q$ of interface automata P and Q is the interface automaton $\pi^I_w(P \otimes Q)$ defined by input strategy π^I_w for the game $P \otimes Q$, where π^I_w is the most permissive winning input strategy, if one exists.

Proposition 1. *For composable interface automata P and Q, $P \parallel Q$ can be obtained as $\pi^I_w(P \otimes Q)$ where π^I_w is the most permissive winning input strategy for $P \otimes Q$. If no winning input strategy exists then P and Q are incompatible.*

Proof. By Definition 13, if no winning input strategy exists, there exists an output strategy π^O such that an incompatible state appears in some sequence $\sigma \in \text{Outcomes}_{P\otimes Q}(s^0_{P\otimes Q}, \pi^I, \pi^O)$, for all input strategies π^I. From Definition 10, this implies that the set T of backward compatible states is empty, and hence by Definition 11 the composition $P \parallel Q$ is empty. Suppose there is a winning input strategy for $P \otimes Q$. We show that the set of states $S_{\pi^I_w(P\otimes Q)}$ is identical with the backward compatible states T of $P \otimes Q$, where π^I_w is the most permissive winning input strategy for $P \otimes Q$. Suppose $s \in S_{\pi^I_w(P\otimes Q)}$. Since π^I_w is a winning strategy, s satisfies the first clause in Definition 10 of backward compatibility. By Definition 15, s is reached in some play in $\text{Outcomes}_{P\otimes Q}(s^0_{P\otimes Q}, \pi^I_w, \pi^O)$ and therefore s satisfies the second clause as well. Now suppose s is a backward compatible state of $P \otimes Q$. By Definition 10 there exists a winning input strategy π^I and some output strategy π^O for $P \otimes Q$, for which s appears in some play $\sigma \in \text{Outcomes}_{P\otimes Q}(s^0_{P\otimes Q}, \pi^I, \pi^O)$. It follows that s appears in some play in $\text{Outcomes}_{P\otimes Q}(s^0_{P\otimes Q}, \pi^I_w, \pi^O)$, and by Definition 15, s is in $S_{\pi^I_w(P\otimes Q)}$. □

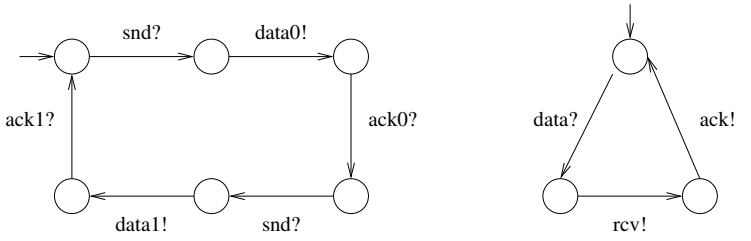
Next we characterise solutions to $P \parallel R \preceq Q$ in terms of winning strategies for Input in $(P \otimes Q^\perp)^\perp$, and show that the most general solution arises from the most permissive strategy.

Theorem 3. *A solution to $P \parallel R \preceq Q$ exists iff a winning input strategy π exists for $(P \otimes Q^\perp)^\perp$. The most general solution to $P \parallel R \preceq Q$ is given by $\pi^I_w((P \otimes Q^\perp)^\perp)$, where π^I_w is the most permissive winning input strategy.*

Proof. From Theorems 1 and 2 it follows that a solution exists iff P and $Q^\perp$ are compatible, and in such a case $R = (P \parallel Q^\perp)^\perp$ is the most general solution. By Proposition 1, $(P \parallel Q^\perp)^\perp = \pi^I_w((P \otimes Q^\perp)^\perp)$ where π^I_w is the most permissive winning strategy for $(P \otimes Q^\perp)^\perp$. □

5 Application: Network Protocol Conversion

In this section we describe an application of interface synthesis to the protocol conversion problem. In today's world global communication over heterogeneous networks of computers can often lead to protocol mismatches between communicating entities. The lack of a uniform global standard for communication protocols entails that protocol converters have to be built for mediating between incompatible protocols [5, 11]. We illustrate the use of interface synthesis to the protocol conversion problem through an example adapted from [10].



(a) Alternating Bit Sender (b) Nonsequenced Receiver

Fig. 2. Two mismatched protocols

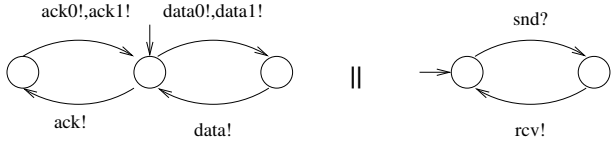


Fig. 3. Specification of Converter

Consider the two interface shown in Figure 2 representing two incompatible protocols. Figure 2(a) is a simplified version of a sender using the Alternating Bit Protocol (ABP), while the one in Figure 2(b) is a receiver using the Nonsequenced Protocol (NS). The ABP sender accepts data from the user (a higher level protocol) using the input action `snd?` and transmits it with label 0 using output action `data0!`. After receiving an acknowledgement with the correct label 0 via the input action `ack0?`, the sender is ready to accept the next piece of data from the user and transmit it with label 1. The protocol performs in a loop, alternating labels between 0 and 1. In this simplified version we ignore retransmissions due to timeouts and receipt of acknowledgements with wrong labels.

The NS receiver in Figure 2(b) is much simpler, which on receiving a data packet via input action `data?`, delivers it to the user via the output action `rcv!`, and sends an acknowledgement to the sender via `ack!`. Since the NS receiver does not use any labels for the data and acknowledgement packets there is a protocol mismatch between ABP and NS.

When we want the two protocols above to work together without causing any inconsistency by using a converter, we need to specify what the converter is allowed and not allowed to do. This idea was proposed in [15] in the setting of synchronous hardware-like protocols. We require that the system as a whole (the two protocols along with the converter) satisfies the interface described by Figure 3. This specification interface is obtained as the parallel composition of two interfaces. The one on the left specifies that the converter can send data packets and acknowledgements to the NS receiver and ABP sender, only after receiving a data packet or acknowledgement from the other protocol. No data or acknowledgement can be sent speculatively, nor can packets be lost or duplicated.

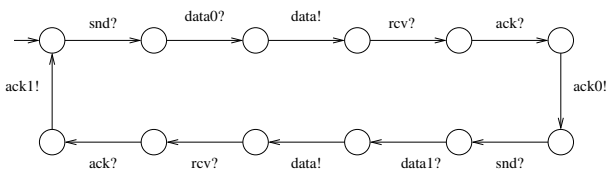


Fig. 4. Converter for the two protocols

The interface on the right specifies the overall behaviour that the user expects from the system: the `snd` and `rcv` events will alternate strictly in any system run. Note that every action in Figure 3 is of type output.

The correct converter for the two protocols is shown is Figure 4. The converter can be obtained be as follows. Let P be the parallel composition of the two protocols which need conversion. Since we assume the two sets of actions to be disjoint, the composition is always well defined. The specification S for the converter relates the two actions sets by specifying temporal ordering of actions. For instance, in our example, the specification dictates that a `data` action can only follow a corresponding `data0` or `data1` action. The converter C is then the (most general) solution to $P \parallel C \preceq S$. Intuitively, the goal of the converter is to meet the specification, while satisfying the input assumptions of the two protocols. Moreover, the converter can control only the inputs to the protocols and not their outputs.

6 Conclusion

We have pointed out the connection between the most general solution to $P \parallel R \preceq Q$ and language equation solving [20, 21], protocol converter synthesis [5] and the semantics of interaction [1] in Section 1. This suggests an underlying algebraic framework for interface automata that is yet to be explored. Such a framework would have axioms and rules for combining interface automata using composition, alternating refinement and $(\_)^{\perp}$. This will simplify the kind of proofs we have presented in Section 3 and Section 4.

Tabuada [18] has shown the connection between control synthesis and the existence of certain alternating simulations and bisimulations between the specification and the system to be controlled. This was carried out using the span of open maps of Joyal *et al* [9]. It would be illuminating to see whether our synthesis problem can be cast in the same framework. To do this, we need to characterise the composition operation $P \parallel Q$ from the product $P \times Q$ in a suitable category of interface automata. Note that it is in the definition of composition that the interface automata formalism differs from the ones considered in [18].

In summary, our work should be seen as a first step towards a unified theory of component interfaces and their synthesis, with wide ranging applications across diverse domains.

Acknowledgements We thank David Benson, Paddy Krishnan, Prahlad Sampath and S. Ramesh for their discussions and critical comments on earlier drafts of the paper.

References

1. S. Abramsky. Semantics of interaction: an introduction to game semantics. In *Proceedings of the 1996 CLiCS Summer School*, pages 1–31. Cambridge University Press, 1997.
2. R. Alur, T.A. Henzinger, O. Kupferman, and M.Y. Vardi. Alternating refinement relations. In *CONCUR 98: Concurrency Theory*, Lecture Notes in Computer Science 1466, pages 163–178. Springer-Verlag, 1998.
3. Andreas Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220, 1992. Special Volume dedicated to the memory of John Myhill.
4. J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.
5. K. L. Calvert and S. S. Lam. Formal methods for protocol conversion. *IEEE Journal Selected Areas in Communications*, 8(1):127–142, January 1990.
6. L. de Alfaro and T.A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*, pages 109–120. ACM Press, 2001.
7. Luca de Alfaro. Game models for open systems. In *Proceedings of the International Symposium on Verification (Theory in Practice)*, volume 2772 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.
8. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
9. André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, June 1996.
10. Ratnesh Kumar, Sudhir Nelvagal, and Steven I. Marcus. A discrete event systems approach for protocol conversion. *Discrete Event Dynamic Systems*, 7(3):295–315, June 1997.
11. S. S. Lam. Protocol conversion. *IEEE Transactions on Software Engineering*, 14(3):353–362, March 1988.
12. Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, 10–12 August 1987.
13. P. Madhusudan and P. S. Thiagarajan. Controllers for discrete event systems via morphisms. In *CONCUR '98: Concurrency Theory, 9th International Conference*, volume 1466 of *Lecture Notes in Computer Science*, pages 18–33. Springer-Verlag, 1998.
14. Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *12th Annual Symposium on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242, Munich, Germany, 2–4 March 1995. Springer.
15. Roberto Passerone, Luca de Alfaro, T.A. Henzinger, and Alberto L. Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: Two faces of the same coin. In *ICCAD '02: Proceedings of the International Conference on Computer Aided Design*, pages 132–140. ACM, 2002.
16. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL '89. Proceedings of the sixteenth annual ACM symposium on Principles of programming*

languages, January 11–13, 1989, Austin, TX, pages 179–190, New York, NY, USA, 1989. ACM Press.

17. P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77, 1:81–98, 1989.
18. Paulo Tabuada. Open maps, alternating simulations and control synthesis. In *CONCUR '04*, number 3170 in Lecture Notes in Computer Science, pages 466–480. Springer-Verlag, 2004.
19. Wolfgang Thomas. On the synthesis of strategies in infinite games. In *12th Annual Symposium on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13, Munich, Germany, 2–4 March 1995. Springer.
20. Nina Yevtushenko, Tiziano Villa, Robert K. Brayton, Alex Petrenko, and Alberto Sangiovanni-Vincentelli. Solution of parallel language equations for logic synthesis. In *Proceedings of the 2001 International Conference on Computer-Aided Design (ICCAD-01)*, pages 103–111, Los Alamitos, CA, November 4–8 2001. IEEE Computer Society.
21. Nina Yevtushenko, Tiziano Villa, Robert K. Brayton, Alex Petrenko, and Alberto Sangiovanni-Vincentelli. Solution of synchronous language equations for logic synthesis. In *Proceedings of the 4th Conference on Computer-Aided Technologies in Applied Mathematics*, pages 132–137, September 2002.