Synthesis of Interface Automata

Purandar Bhaduri

Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati,
Guwahati 781039, India
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 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 $P \parallel R \preceq P$ for $R$ exists for deterministic interface automata iff $P$ and $Q$ are compatible, and the most abstract (under alternating refinement) solution is given by $(P \parallel Q)^\perp$. Further, such an $R$ can be constructed from the most permissive winning strategy for player Input in the combined game $(P \otimes Q)^\perp$. Here $P^\perp$ is the game $P$ with the moves of the players Input 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 $\pi$ is more permissive than $\pi'$ when, at every position in the game, the set of moves allowed by $\pi$ includes those allowed by $\pi'$. 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 $(P \otimes Q^\perp)^\perp = P^\perp \triangleright Q = P \multimap Q$, where $\otimes$, $\triangleright$ 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 $\parallel$. The striking similarity of this solution with the language equation posed in