# Submodule Construction for Specifications with Input Assumptions and Output Guarantees *

Gregor v. Bochmann

School of Information Technology and Engineering (SITE), University of Ottawa, Canada
bochmann@site.uottawa.ca

**Abstract.** We consider the following problem: For a system consisting of two submodules, the behavior of one submodule is known as well as the desired behavior S of the global system. What should be the behavior of the second submodule such that the behavior of the composition of the two submodules conforms to S? – Solutions to this problem have been described in the context of various specification formalisms and various conformance relations. Here we present a generalization of this problem and its solution in the context of relational databases, and show that this general solution can be used to derive several of the known algorithms that solve the problem in the context of regular behavior specifications based on finite state automata with synchronous communication or interleaving semantics. The paper also provides a new solution formula for the case that the module behaviors are specified in a hypothesis-guarantee paradigm distinguishing between input and output. A new submodule construction algorithm for synchronous, partially defined input/output automata is also given.

## 1 Introduction

In automata theory, the notion of constructing a product machine S from two given finite state machines S1 and S2, written S = S1 x S2, is a well-known concept. This notion is very important in practice since complex systems are usually constructed as a composition of smaller subsystems, and the behavior of the overall system is in many cases equal to the composition obtained by calculating the product of the behaviors of the two subsystems. Here we consider the inverse operation, also called equation solving: Given the composed system S and one of the components S1, what should be the behavior S2 of the second component such that the composition of these two components will exhibit a behavior equal to S. That is, we are looking for the value of X which is the solution to the equation S1 x X = S. In fact, we are looking for the most general machine X which composed with S1 satisfies some conformance relation in respect to S. In the simplest case, this conformance relation is trace inclusion.

A first paper of 1980 [Boch 80d] (see also [Merl 83]) gives a solution to this problem for the case where the machine behavior is described in terms of labeled transi-

* This work was partly supported by a research grant from the Natural Sciences and Engineering Research Council of Canada. This paper was written when the author was a visiting professor at Osaka University, Japan.

D.A. Peled and M.Y. Vardi (Eds.): FORTE 2002, LNCS 2529, pp. 17–33, 2002.

tion systems (LTS) which communicate with one another by synchronous interactions (see also [Hagh 99] for a more formal treatment). This work was later extended to the cases where the behavior of the machines is described in CCS or CSP (with behavioral equivalence as conformance relation) [Parr 89], by finite state machines (FSM) communicating through message queues [Petr 98, Yevt 01a], by input/output automata [Dris 99a] ([Qin 91] considers bisimulation as conformance relation), and by synchronous finite state machines [Kim 97]. A restricted version of this problem is considered in an earlier paper [Kim 72] which considers series composition of FSMs where each message goes from the originating FSM to its neighbor on the right (no feedback). The specification of the rightmost FSM is derived from the behavior of the other FSMs and the desired behavior of the composition.

For a discussion of the applications of this equation-solving method in communication protocol design and control theory, we refer the reader to [Boch 02a].

In this paper we show that the above equation solving problem in the different contexts of LTS, synchronous and asynchronous FSMs and IOA are all special cases of a more general problem which can be formulated in the context of relational database theory which is generalized to allow for non-finite relations (i.e. relations representing infinite sets). We give the solution of this general problem and give a proof of its correctness. We also show how the different specialized version of this problem – and the corresponding solutions – can be derived from the general database version.

After a review of basic notions of relational databases, we present in Section 3 the problem of equation solving in the database context and provide solution formulas and their proofs. In Section 4, we discuss how the database model can be adapted to model the dynamic behavior of systems and their components based on trace semantics, that is, when the behavior of a system component is characterized by the set of possible traces of interactions in which it could participate. We consider the cases of synchronous rendezvous communication and interleaving semantics. We also explain how the solution formula for databases can be used to derive solution algorithms for systems with regular behavior (i.e. described by finite state transition systems). In Section 5 we introduce the distinction of input and output which allows the specification of a component behavior using the hypothesis-guarantee paradigm. We state appropriate conformance relations which can be used to define the submodule construction problem. We present a general solution formula which leads to several submodule construction algorithms that can be applied to different variants of regular behavior specifications, including a variant allowing for nondeterministic output and partially defined behaviors.

## 2 Review of Some Notions from the Theory of Relational Databases

The following concepts are defined in the context of the theory of relational databases [Maie 83]. Informally, a relational database is a collection of relations where each relation is usually represented as a table with a certain number of columns. Each column corresponds to an attribute of the relation and each row of the table is called a tuplet. Each tuplet defines a value for each attribute of the relation. Such a tuplet represents usually an “object”, for instance, if the attributes of the *employee* relation are

*name*, *city*, *age*, then the tuplet <Alice, Ottawa, 25> represents the employee "Alice" from "Ottawa" who is 25 years old.

The same attribute may be part of several relations. Therefore we start out with the definition of all attributes that are of relevance to the system we want to describe.

**Definition** ***(attributes and their values):*** The set $A = \{a_1, a_2, \ldots, a_m\}$ is the set of attributes. To each attribute $a_i$ is associated a (possibly infinite) set $D_i$ of possible values that this attribute may take. $D_i$ is called the domain of the attribute $a_i$. We define $D = U\ D_i$ to be the discriminate union of the $D_i$ .

**Definition** ***(relation):*** Given a subset $A_r$ of A, a relation R over $A_r$, written $R[A_r]$, is a (possibly infinite) set of mappings T: $A_r$ --> D with $T(a_i)\ \varepsilon\ D_i$. An **integrity constraint** is a predicate on such mappings. If the relation R has an integrity constraint C, this means that for each $T \varepsilon R$, C(T) is true.

Note: In the informal model where a relation is represented by a table, a mapping T corresponds to a tuplet in the table. Here we consider relations that may include an infinite number of different mappings.

**Definition** ***(projection):*** Given $R[A_r]$ and $A_x \subseteq A_r$ , the projection of $R[A_r]$ onto $A_x$, written $proj_{Ax}$ (R), is a relation over $A_x$ with

$T \varepsilon proj_{Ax}$ (R) iff there exists T' $\varepsilon$ R such that for all $a_i \varepsilon A_x$ , $T(a_i) = T'(a_i)$

We note that T is the restriction of T' to the subdomain $A_x$. We also write $T = proj_{Ax}$ (T').

**Definition** ***(natural join):*** Given $R_1[A_1]$ and $R_2[A_2]$, we define the (natural) join of the relations $R_1$ and $R_2$ to be a relation over $A_1\ U\ A_2$ , written $R_1$ join $R_2$ , with

$T \varepsilon (R_1$ join $R_2)$ iff $proj_{A1}$ (T) $\varepsilon R_1$ and $proj_{A2}$ (T) $\varepsilon R_2$

**Definition** ***(chaos):*** Given $A_r \subseteq A$, we call chaos over $A_r$ , written $Ch[A_r]$ , the relation which includes all elements T of $A_r$ --> D with $T(a_i)\ \varepsilon\ D_i$ , that is, the union of all relations over $A_r$.

Note: We note that $Ch[A_r]$ is isomorphic to the Cartesian product of the domains of all the attributes in $A_r$ . The notion of "chaos" is not common in database theory. It was introduced by Hoare [Hoar 85] to denote the most general possible behavior of a module. It was also used in several papers on submodule construction [Petr 98, Dris 99a].

It is important to note that we consider here infinite attribute value domains and relations that contain an infinite number of mappings (tuplets). In the context of traditional database theory, these sets are usually finite (although some results on infinite databases can be found in [Abit 95]). This does not change the form of our definitions, however. If one wants to define algorithms for solving equations involving such infinite relations, one has to worry about the question of what kind of finite representations should be adopted to represent these relations. The choice of such representations will determine the available algorithms and at the same time introduce restrictions on the generality of these algorithms. Some of these representation choices are considered in Sections 4 and 5.

## 3 Equation Solving in the Context of Relational Databases

We consider here a very simple configuration with three attributes $a_1$, $a_2$, and $a_3$, and three relations $R_1[\{a_2, a_3\}]$, $R_2[\{a_1, a_3\}]$, and $R_3[\{a_2, a_1\}]$ as shown in the figure below. A more general architecture has been considered in [Boch 02a].

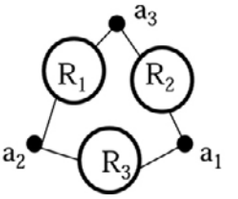


**Fig. 3.1.** Configuration of 3 relations sharing 3 attributes

We consider the following equation (which is in fact an inclusion relation)

$$\text{proj}_{\{a2, a1\}} (R_1 \text{ join } R_2) \subseteq R_3 \qquad \text{(Equ. 1)}$$

If the relations $R_1$ and $R_3$ are given, we can ask the question: for what relation $R_2$ will the above equation be true. Clearly, the empty relation, $R_2 = \bullet$ (empty set), satisfies this equation. However, this case is not very interesting. However, we note that there is always a single maximal solution. This solution is the set

$$Sol^{(2)} = \{T \in Ch[\{a_1, a_3\}] \mid \text{proj}_{\{a2, a1\}} (R_1 \text{ join } \{T\}) \subseteq R_3 \} \qquad \text{(Equ. 2)}$$

This is true because the operators of set union and intersection obey the distributive law in respect to the join operation, that is, $R_i$ join ($R_j$ union $R_k$) = ( $R_i$ join $R_j$) union ( $R_i$ join $R_k$); and similarly for intersection.

While the above characterization of the solution is trivial, the following formula is useful for deriving algorithms that obtain the solution in the context of the specific representations discussed in Sections 4 and 5.

**Theorem:** A solution for $R_2$ that satisfies Equation (1), given $R_1$ and $R_3$ , is given by the following formula (where "/" denotes set substraction):

$$Sol^{(3)} = Ch[\{a_1, a_3\}] / \text{proj}_{\{a1, a3\}} ( R_1 \text{ join } ( Ch[\{a_1, a_2\}] / R_3 ) ) \qquad \text{(Equ. 3)}$$

This is the largest solution and all other solutions of Equ. (1) are included in this one.

Informally, Equation (3) means that the largest solution consists of all tuplets over $\{a_1, a_3\}$ that cannot be obtained from a projection of a tuplet T $[\{a_1, a_2, a_3\}]$ that can be obtained by a join from an element of $R_1$ and a tuplet from $Ch[\{a_1, a_2\}]$ that is not in $R_3$.

We note that the smaller solution

$$Sol^{(3*)} = \text{proj}_{\{a1, a3\}}( R_1 \text{ join } R_3) / \text{proj}_{\{a1, a3\}} (R_1 \text{ join } ( Ch[\{a_1, a_2\}] / R_3 ) ) \quad \text{(Equ. 3*)}$$

is also an interesting one, because it contains exactly those tuplets of $Sol^{(3)}$ that can be joint with some tuplet of $R_1$ to result in a tuplet whose projection on $\{a_1, a_2\}$ is in $R_3$. Therefore ($R_1$ join $Sol^{(3)}$) and ($R_1$ join $Sol^{(3*)}$) are the same set of tuplets; that means the same subset of $R_3$ is obtained by these two solutions. In this sense, these solutions are

equivalent. We note that the solution formula given in [Merl 83] corresponds to the solution $Sol^{(3*)}$.

**Proof of the theorem:** First we note that ($T_2$ ε $Sol^{(3)}$ ) is equivalent to the statement that there exist no T ε Ch [{$a_1$, $a_2$, $a_3$}] such that

$$proj_{\{a1, a3\}}(T) = T_2 \textbf{ and } proj_{\{a2, a3\}}(T) \varepsilon R_1 \textbf{ and } proj_{\{a1, a2\}}(T) \neg\varepsilon R_3$$

We call this **(Equivalence 4)**.

We have to prove that $Sol^{(3)} = Sol^{(2)}$ . In order to show that $Sol^{(3)} \subseteq Sol^{(2)}$, we show that

$$proj_{\{a2, a1\}} (R_1 \text{ join } Sol^{(3)}) \subseteq R_3 \qquad \text{(Equ. 5)}$$

Taking any T' ε ($R_1$ join $Sol^{(3)}$), we have $proj_{\{a2, a3\}}(T') \varepsilon R_1$ and $proj_{\{a1, a3\}}(T') \varepsilon Sol^{(3)}$ . Since $proj_{\{a1, a3\}}(T') \varepsilon Sol^{(3)}$ , there is, according to Equivalence (4), no T ε Ch [{$a_1$, $a_2$, $a_3$}] such that $proj_{\{a1, a3\}}(T) = proj_{\{a1, a3\}}(T')$ and $proj_{\{a2, a3\}}(T) \varepsilon R_1$ and $proj_{\{a1, a2\}}(T) \neg\varepsilon R_3$ . Since T' satisfies the first two of these three conditions, we conclude that the last condition must be false for T'. Therefore we have that $proj_{\{a1, a2\}}(T') \varepsilon R_3$ which implies Equation (5).

In order to prove that $Sol^{(3)} \supseteq Sol^{(2)}$ , we assume that this is not true and that there exist a tuplet T' that is in $Sol^{(2)}$ , but not in $Sol^{(3)}$ . However, the latter implies, according to Equivalence (4), that there exists a T ε Ch [{$a_1$, $a_2$, $a_3$}] such that

$$proj_{\{a1, a3\}}(T) = T' \textbf{ and } T_1 =^{def} proj_{\{a2, a3\}}(T) \varepsilon R_1 \textbf{ and } proj_{\{a1, a2\}}(T) \neg\varepsilon R_3 \qquad \text{(Equ. 6)}$$

Considering the definition of the join operation, we conclude that {T} = {T'} join {$T_1$} since the join of two singleton relations contains at most one tuplet. But now we have a contradiction because ( T' ε $Sol^{(2)}$ ) implies $proj_{\{a1, a2\}}$ ({T'} join {$T_1$}) ⊆ $R_3$ while Equation (6) states $proj_{\{a1, a2\}}$ ({T'} join {$T_1$}) ¬ε $R_3$ . Therefore our assumption must be false. Q.E.D.

# 4 Equation Solving in the Context of Composition of Sequential Machines or Reactive Software Components

## 4.1 Modeling System Components and Behavior Using Traces

Sequential machines and reactive software components are often represented as black boxes with *ports*. The ports are the places where the *interactions* between the component in question and the components in its environment take place.

For allowing the different modules to communicate with one another, their ports must be interconnected. Such interconnection points are usually called *interfaces*. An example of a composition of three modules (sequential machines or reactive software components) is shown in Figure 4.1. Their ports are pair-wise interconnected at three interfaces $a_1$, $a_2$, and $a_3$.

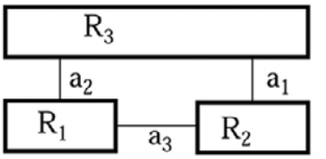


**Fig. 4.1.** Configuration of 3 components interconnected through 3 interfaces

The dynamic behavior of a module (sequential machine or a reactive software component) is usually described in terms of *traces*, that is, sequences of interactions that take place at the interfaces to which the module is connected. Given an interconnection structure of several modules and interfaces, we define for each interface i the set of possible interactions $I_i$ that may occur at that interface. For each (finite) system execution trace, the sequence of interactions observed at the interface $a_i$ is therefore an element of $I_i$ * ( a finite sequence of elements in $I_i$ ).

For communication between several modules, we consider in this paper rendezvous interactions. This means that, for an interaction to occur at an interface, it is necessary that all modules connected to that interface must make a state transition compatible with that interaction at that interface.

In our basic communication model we assume that the interactions between the different modules within the system are synchronized by a clock, and that there must be an interaction at each interface during each clock period. We call this "synchronous operation".

## 4.2 Correspondence with the Relational Database Model

We note that the above model of communicating system components can be described in the formalism of (infinite) relational databases as follows:

1. A port corresponds to an attribute and a module to a relation. For instance, the interconnection structure of Figure 4.1 corresponds to the relationship shown in Figure 3.1. The interfaces $a_1$, $a_2$, and $a_3$ in Figure 4.1 correspond to the three attributes $a_1$, $a_2$, and $a_3$ in Figure 3.1, and the three modules correspond to the three relations.
2. If a given port (or interface) corresponds to a particular attribute $a_i$, then the possible execution sequences $I_i$* occurring at that port correspond to the possible values of that interface, i.e. $D_i = I_i$* .
3. The behavior of a module $M_x$ is given by the tuplets $T_x$ contained in the corresponding relation $R_x$ [$A_x$], where $A_x$ corresponds to the set of ports of $M_x$. That is, a trace $t_x$ of the module X corresponds to a tuplet $T_x$ which assigns to each interface $a_i$ the sequence of interactions $s_{xi}$ observed at that interface during the execution of this trace. We write $s_{xi}{}^{@t}$ to denote the t-th element of $s_{xi}$

Since we assume "synchronous operation" (as defined in Section 4.1), all tuplets in a relation describing the behavior of a module must satisfy the following constraint:

**Synchrony constraint:** The length of all attribute values are equal. (This is the length of the trace described by this tuplet.)

### 4.3 The Case of Synchronous Finite State Machines

If we restrict ourselves to the case of regular behavior specifications, where the (infinite) set of traces of a module can be described by a finite state transition model, we can use Equation (3) or Equation (3*) to derive an algorithm for equation solving. In this case, the behavior specification for a module is given in the form of a finite state automaton (similar to labeled transition systems) where each transition is labeled by a set of interactions, one for each port of the module. We note that the synchronous composition considered here is different than the synchronous composition of Mealy or Moore machines, as considered in [Kim 97] since the latter distinguish between input and output, as discussed in Section 5. (It appears that the complete version of [Yevt 01a] also deals with this synchronous case).

The algorithm for equation solving is obtained from Equation (3) or Equation (3*) by replacing the relational database operators *projection*, *join* and *substraction* by the corresponding operations on finite state automata. The database *projection* corresponds to eliminating those interaction labels from all transitions of the automaton which correspond to attributes that are not included in the set of ports onto which the projection is done. This operation, in general, introduces nondeterminism in the resulting automaton. The *substraction* operation is of linear complexity if its two arguments are deterministic finite state automata. Since the projection operator introduces nondeterminism, one has to include a step to transform the nondeterministic automata obtained from the *projection* into its equivalent deterministic form. This step is in general of exponential complexity. However, our experience with some examples involving the interleaved semantics described below [Dris 99a] indicates that reasonably complex systems can be handled in many cases. The well-known algorithm for performing this step consists of building a deterministic automata which has as its states the possible subsets of states (of the original automaton) that are reachable after a given trace of interactions.

In fact, the substraction operations involved in Equation (3) are of a special form; they represent the construction of the complement. This is a simple operation for a deterministic automaton. In a first step, the automaton is completed, that is, an additional (non-accepting) state, called *fail* state, is introduced, and from each state of the automaton additional transitions are created from the state to the *fail* state for all those interaction labels for which the original automaton has no transition from that state. There is also a self-loop on the *fail* state for all possible interaction labels. This first step does not change the traces of interactions accepted by the automaton. The second step performs the complement by exchanging the accepting and non-accepting states.

The *join* operation corresponds to the composition operator of automata which is of polynomial complexity. In the case of the synchronous operation considered here, the composition is defined as follows.

**Synchronous composition:** Given two automata R1 [a1, a3] and R2 [a2, a3] with sets of states S1 and S2, respectively, and transitions written $s_i$ – x, y --> $s_i$' with $s_i$ and $s_i$' ε Si, for i = 1, 2, and where x and y are interactions at the two ports of the automaton in question. The synchronous composition of these two automaton is an automaton R [a1, a2, a3] which is defined as follows:

- The states of R are of the form ($s_1$, $s_2$) where $s_1$ ε S1 and $s_2$ ε S2; the initial state is the pair of the initial states of the R1 and R2, and a state ($s_1$, $s_2$) is accepting if $s_1$and $s_2$ are accepting in their respective automata.

- R3 includes a transition $(s_1, s_2)$ – x, y, z --> $(s_{1'}, s_{2'})$ iff the transitions $s_1$ – x, z --> $s_1$' and $s_2$ – y, z --> $s_2$' exist in their respective automata.

The composition of two automata can be easily constructed by starting from the initial state and constructing all states of the composition that are reachable.

Figure 4.2 shows a simple example. $R_3$ and $R_1$ are given. The note "Notation $(x_1,x_2)$" for $R_3$ means that the transition labels of $R_3$ first contain the interaction at interface $a_1$ and then at $a_2$. The submodule construction algorithm proceeds as follows: First R3 is completed with the introduction of a *fail* state, then the product with R1 is constructed. When the interations at the interface a2 are projected out, there is nondeterminism in state (2,2) for the label (n,n,d). This leads to a determinized automaton. From this automaton, all traces leading to a state containing *fail* should be eliminated. This leads to the elimination of the transition (n,n) from state ((4,3), (5,3)), but then also the transition leading to this state is eliminated since this state represents a deadlock. (Deadlock elimination is not further considered here).

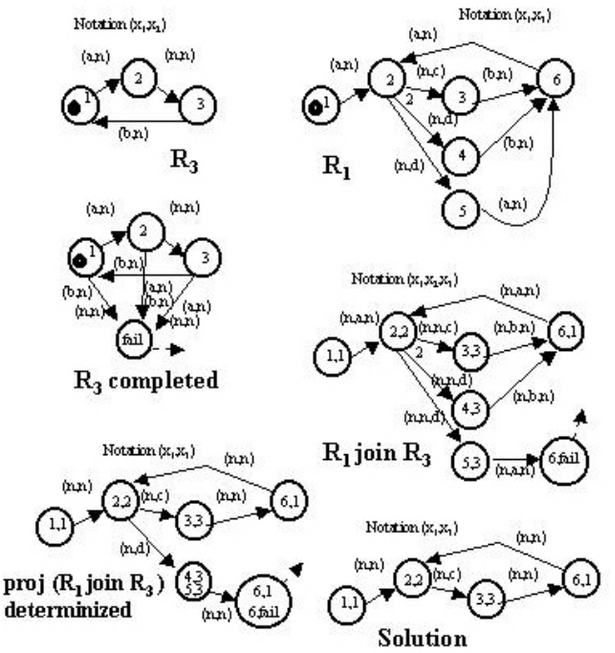


**Fig. 4.2.** Example of submodule construction for synchronous automata

## 4.4 The Case of Interleaving Rendezvous Communication

In this subsection, we consider non-synchronous rendezvous communication also called interleaving semantics, were at each instant in time at most one interaction takes place within all interconnected system components. This communication paradigm is used, for instance, with labeled transition systems (LTS), CSP and LOTOS.

One way to model the behavior of such systems is to consider a global execution trace which is the sequence of interactions in the order in which they take place at the different interfaces (one interface at a time). Each element of such an execution sequence defines the interface $a_i$ at which the interaction occurs and the interaction $v_i$ which occurs at that interface.

Another way to represent the behavior of such systems is to reduce it to the case of synchronous communication as follows. This is the approach which we adopt in this paper because it shows how the interleaving rendezvous communication can be based on our relational database model. In order to model the interleaving semantics, we postulate that all sets $I_i$ include a dummy interaction, called *null*. It represents the fact that no interaction takes place at the interface. We then postulate that each tuplet T of a relation R[A] satisfies the following constraint:

**Interleaving constraint:** For all time instants t (t > 0) we have that $T(a_i)[t] \bullet null$ implies $T(a_j)[t] = null$ for all $a_j \varepsilon A$ $(j \bullet i)$

We note that tuplets that are equal to one another except for the insertion of time periods during which all interfaces have the *null* interaction are equivalent as far as the sequence of non-*null* interactions is concerned. Note that this equivalence is sometimes called stuttering equivalence. In the following we are only interested in equivalent classes in respect to this stuttering equivalence. Such a class may be represented by the interaction sequence in the class that has no time instance with *null* interactions at **all** interfaces. We say that such an interaction sequence is of "normal form". We also assume that each relation contains, with each sequence of interactions, also all other sequences that are stuttering equivalent to it. We call this the **stuttering completeness assumption**.

The relational database operators *projection* and *substraction* apply under the interleaving constraint in the normal way. However, the result obtained by the joining of two relations satisfying the interleaving constraint may include tuplets that do not satisfy this constraint, because the joint interaction sequences may have non-*null* interactions at the same time at different interfaces. We assume that such sequences not satisfying the interleaving constraint will be eliminated from the original result of the join. However, because of the stuttering completeness assumption, there will also be an interaction sequence in the original join results which contains the conflicting interactions at different time instants. Therefore the result of a join will include all the interleavings of the non-*null* interactions as far as they are conform with each of the joint relations. We can therefore conclude that the interleaving semantics defined here corresponds exactly to the well-known interleaving semantics of labeled transition systems.

### 4.5 The Case of Finite Labeled Transition Systems

To simplify the notation, we assume that the sets of interactions at different interfaces are disjoint (i.e. $I_i$ intersection $I_j$ = empty for $a_i \bullet a_j$), and we introduce the overall set of interactions $I = U_{(ai\ \varepsilon\ A)}\ I_i$. Then a class of stuttering equivalent interleaving traces (as described in Section 4.4) corresponds one-to-one to a sequence of interactions in I.

If we restrict ourselves to the case where the possible traces of a module are described by a finite LTS, the resulting set of possible execution sequences are regular

sets. In fact, a finite LTS is a finite state automaton with only accepting states. Therefore the operations *projection*, and *substraction* over interleaving traces can be represented by finite operations over the corresponding automata representations, as in the case of synchronous operation discussed in Section 4.4. Again, the projection may introduce nondeterminism (in the form of spontaneous transitions, sometimes written with label i) and a determination step is required before the substraction operation. Finally, the join operation represents the composition of LTSs and is defined as follows:

**Interleaved composition:** Given two automata R1 [a1, a3] and R2 [a2, a3] with sets of states S1 and S2, respectively, and transitions written $s_i$ – x --> $s_i$' with $s_i$ and $s_i$' ε Si, for i = 1, 2, and where x is an interaction at one of the two ports of the automaton in question. The interleaved composition of these two automaton is an automaton R [a1, a2, a3] which is defined as follows:

- The states of R are of the form ($s_1$, $s_2$) as for synchronous composition.
- R3 includes a transition ($s_1$, $s_2$) – x --> ($s_{1'}$, $s_{2'}$) iff one of the following conditions is satisfied:
  - x ε I1 and $s_1$ – x --> $s_{1'}$ and $s_2$ = $s_{2'}$
  - x ε I2 and $s_2$ – x --> $s_{2'}$ and $s_1$ = $s_{1'}$
  - x ε I3 and $s_1$ – x --> $s_{1'}$ and $s_2$ – x --> $s_{2'}$

The submodule construction algorithm defined by Equation (3*) based on the operations on automata described above is equal to the construction algorithm that we described earlier [Boch 80d, Merl 83]. The proof of correctness of Equation (3) given in Section 3 also provides a proof of this algorithm.

The example of Figure 4.2 may be considered as an example for interleaving semantics. In fact, the specification has the particular form that, if we consider the interaction written "n" as the *null* interaction, then all the specifications in the figure satisfy the interleaving constraint.

# 5 Distinction of Input and Output

## 5.1 Module Specification Based on Hypothesis and Guarantees

The rendezvous communication paradigm considered in Section 4 has a drawback when it comes to its use for requirements specification. Usually, the requirements for a system module has two parts: (a) the hypothesis that the module may make about the behavior of the other modules within its environment and general operating assumptions such as temperature ranges etc., and (b) the guarantees that the module must provide concerning the behavior it will exhibit during execution.

The distinction between these two aspects cannot be made clearly with the rendezvous communication paradigm because for any interaction to occur, it is necessary that all participating modules are ready for it. There is no notion that one of the modules is particularly responsible for initiating the interaction.

We consider in the following a communication paradigm where, for each interaction taking place at some interface, there is one participating module for which the interaction is *output*, and it is *input* for all other modules that are connected to that

interface. Whether the interaction will take place or not, and what its parameters will be, will solely be determined by the outputting module (the interaction must satisfy the guarantees provided by this module). The other participating modules for which the interaction is *input* do not influence the occurrence of the interaction and the values of its parameters. However, they may make the hypothesis that the outputting modules will satisfy the guarantees defined by their respective specifications, thus limiting the range of possibilities for receiving the interaction in question.

This paradigm is the basis for the semantics of (input-output) finite state machines, Input/Output Automata (IOA) [Lync 89], as well as many software specification formalisms, such as [Adab 95, Misr 81]. It seems that this paradigm also subsumes the paradigm of controllable and uncontrollable interactions as considered for discrete event control design [Rama 89]. We note that in the case of finite state machines and IOA, we consider partially defined machines; the *hypothesis* is made that only those inputs will occur for which a transition is defined.

We can introduce the distinction between input and output in our general relational database formalism as follows: Each attribute of a relation is marked as either *input* or *output*. An attribute of a relation resulting from a *join* operation is marked *input* if the same attribute is marked as *input* in the two operands of the *join* operation, otherwise it is marked *output*. A join operation is said to have **output conflict** if there is an attribute that is marked output for both operands. We consider in the following only join operations without output conflict.

We now introduce the following notations. Given a relation $R[A_R]$ and a tuplet $T \varepsilon R$, we write $T^{|t}$ for the tuplet which has as values for an attribute $ai \varepsilon A_R$ the prefix (of length t) of the value which T has for this attribute. For example, if T = <abc, def> then $T^{|2}$ = <ab, de>. And we write $T^{@t}$ for the tuplet which has as value for an attribute $ai \varepsilon A_R$ the t-th element of the sequence which is the value of T for this attribute.For the example of T above, we have $T^{@1}$ = <a, d> and $T^{@3}$ = <c, f>. Similarly, we write $T^{@t}(a_i)$ to denote the t-th element of $T(a_i)$.

In order to clearly distinguish between the input and output attributes of a relation R, we write $R[A_R^I \mid A_R^O]$ where $a_i \varepsilon A_R^I$ are the input attributes of R and $a_i \varepsilon A_R^O$ the attributes marked output.

### 5.2 Conformance Relations

In trace semantics without the distinction of input and output, as discussed in Section 4, the conformance relations are very simple and can be summarized by the following definitions:

1. **Valid trace:** A tuplet (trace) T is valid in respect to a relation (specification) R if $T \varepsilon R$.
2. **Trace inclusion:** A specification R' conforms to a specification R iff all the traces of R' are also valid in respect to R.

In order to define meaningful relations in the context of synchronous operation, we assume that a specification satisfies the constraint that the output allowed at time t by the specification does not depend on the input received at time t (but only on previous inputs and outputs). This implies that a delay of at least one time unit exists between a

received input and the output which is *caused* by this input. The importance of this assumption is discussed in [Abad 95, Broy 95].

In addition, we assume that the hypothesis made by a specification about the validity of the received input at a given time instance does not depend on the output selected by the module at the same time instance. We call these two assumptions together **the unit-delay constraint** (UDC), which can be formally defined as follows:

1. Given a trace specifications $R[A_R]$ and a tuplet $T \varepsilon R$, we write next(T, R) for the relation that describes the possible interactions at the next time instant, formally: $T' \varepsilon$ next(T, R) iff the tuplet T' is of length one and $T.T' \varepsilon R$, where "." denotes the pairwise concatenation of corresponding attribute values.
2. A trace specification (relation) $R[A_R^I \mid A_R^O]$ satisfies the UDC iff for any $T \varepsilon R$ the following holds: next(T, R) = proj $A_R^I$ (next(T, R)) join proj $A_R^O$ (next(T, R))

For characterizing conformance relations, it is important to distinguish different cases of invalid traces. If a given trace (tuplet) T is not valid in respect to a given trace specification (relation) $R[A_R^I \mid A_R^O]$ (i.e. not $T \varepsilon R$), we may consider the longest valid prefix of T; there must exist a time instant $t > 0$ such that $T^{lt\text{-}1} \varepsilon R$ and $T^{@t} \neg\varepsilon$ next($T^{lt\text{-}1}$, R) (we use the notation where $\neg\varepsilon$ means "not included in"). We now can distinguish whether the invalidity of the trace is caused by a wrong input or a wrong output at time instant t as follows:

- **Wrong output:** We say that T has wrong output at time t, written $T \varepsilon R^{WO(t)}$, iff $T^{lt\text{-}1} \varepsilon R$ and proj $A_R^O$ $T^{@t} \neg\varepsilon$ proj $A_R^O$ next($T^{lt\text{-}1}$, R).
- **Wrong input:** We say that T has wrong input at time t, written $T \varepsilon R^{WI(t)}$, iff $T^{lt\text{-}1} \varepsilon R$ and proj $A_R^I$ $T^{@t} \neg \varepsilon$ proj $A_R^I$ next($T^{lt\text{-}1}$, R).

Clearly, it could also happen that T has wrong input **and** wrong output at time t.

Based on the above definitions, we can now formally define the meaning of a component specification $R[A_R^I \mid A_R^O]$ (similar to [Abad 94]) as follows:

1. A trace T over the alphabet $A = A_R^I \cup A_R^O$ **satisfies the guarantees of R**, written $T \text{ sat}_G R$, iff for all $t > 0$ the following holds: $T^{lt\text{-}1} \varepsilon R$ implies $T \neg \varepsilon R^{WO(t)}$.
2. A trace T over A **satisfies the hypotheses of R**, written $T \text{ sat}_H R$, iff for all $t > 0$ the following holds: $T^{lt\text{-}1} \varepsilon R$ implies $T \neg \varepsilon R^{WI(t)}$.
3. A trace T over A **satisfies the specification R**, written T sat R, iff ($T \text{ sat}_H R$) implies ($T \text{ sat}_G R$)
4. A trace T over an arbitrary (larger) alphabet satisfies the specification $R[A_R^I \mid A_R^O]$ iff the projection of T onto $A = A_R^I \cup A_R^O$ satisfies R.
5. Given an interconnection structure containing several components with their respective behavior specifications $R_k$ (i = 1, 2, …, n), we say that a trace T satisfies the interconnection structure iff it satisfies the specifications of all component specifications $R_k$.
6. Another specification $R'[A_R^I \mid A_R^O]$ **conforms to** $\mathbf{R}[A_R^I \mid A_R^O]$ iff for all traces T we have (T sat R') implies (T sat R).

### 5.3 Equation Solving for Specifications with Hypothesis and Guarantees

Taking into account the difference between input and output as discussed above, the problem of equation solving must be formulated in a form different from Equation (1) in Section 3. Now we want to find the most general specification for $R_2$ such that all traces that satisfy the interconnection structure of the modules $R_1$ and $R_2$ (see Figure 5.1), and that also satisfy the hypothesis of $R_3$, have the following two properties: (a) the guarantees of $R_3$ are satisfied, and (b) the hypotheses of $R_1$ are satisfied.

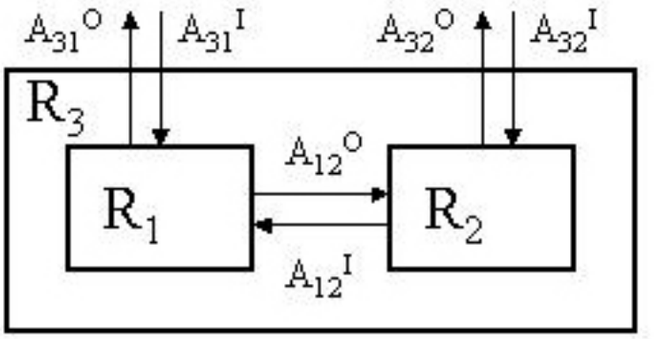


**Fig. 5.1.** Composition of components $R_1$ and X with input/output interactions

This can be formalized as follows. We first note that we consider the alphabet A = $A_{31}^{O}$ U $A_{31}^{I}$ U $A_{32}^{O}$ U $A_{32}^{I}$ U $A_{12}^{O}$ U $A_{12}^{I}$, as shown in the figure. We introduce the following abbreviations for the alphabets of the modules R1, X and R3, respectively:

A1 = $A_{31}^{O}$ U $A_{31}^{I}$ U $A_{12}^{O}$ U $A_{12}^{I}$,
A2 = $A_{32}^{O}$ U $A_{32}^{I}$ U $A_{12}^{O}$ U $A_{12}^{I}$,
A3 = $A_{31}^{O}$ U $A_{31}^{I}$ U $A_{32}^{O}$ U $A_{32}^{I}$ .

We also note that the elements of ($A_{31}^{O}$ U $A_{12}^{O}$) are the outputs of $R_1$, the other elements of A1 are its inputs, $A_{32}^{O}$ U $A_{12}^{I}$ are the outputs of X, the other elements of A2 are its inputs, and $A_{31}^{O}$ U $A_{32}^{O}$ are the outputs of $R_3$, the other elements of A3 are its inputs.

Given two relations $R_1$ and $R_3$ , the equation solving problem, now, consists of finding a set of traces X[A2] which satisfies Equation ($1^{IO}$) below:

$\text{proj}_{A3}(R_1 \text{ join } X)$ conforms to $R_3$ (Equ. $1^{IO}$)

**Theorem**: The set of traces $Sol^{(IO)}$ defined by Equation ($3^{IO}$) is the largest set satisfying Equation ($1^{IO}$):

$$Sol^{(IO)} = Ch[A2] / \text{proj}_{A2}\, U_{t>0}\, ( \quad \text{(Equ. } 3^{IO}\text{)}$$
$$(R_1^{T} \text{ join } R_3^{WO(t)}) \;U\; ((R_1^{T})^{WO(t)} \text{ join } R_3) \;U\; ((R_1^{T})^{WO(t)} \text{join } R_3^{WO(t)}) \;)$$

where the notation $R^{T}$ denotes the relation R with the input/output markings of the ports interchanged.

We note that the traces in $(R_1^{T})^{WO(t)}$ are the same (if one ignores the input/output assignment) as the traces in $R_1^{WI(t)}$, that is, these are the traces that do not satisfy the

hypothesis that $R_1$ makes about the input interactions. The proof of the above theorem is given in [Boch 01b].

### 5.4 The Case of Completely Defined and Deterministic Finite State Machines

Submodule construction for deterministic, completely defined finite state machines is discussed in detail in Chapter 6 of [Kim 97]. Our above assumption of the unit-delay constraint corresponds to the assumption of Moore machines for which the output is a function of the current state. [Kim 97] mainly considers deterministic machines (for which the output is a function of the present state and the input) which are completely defined, that is, in each state and each input there is a specified next state and output; therefore there is no wrong input. Under these assumptions, Equation ($3^{IO}$) becomes

$$\begin{aligned} Sol^{(IO)} &= Ch[A2] / proj_{A2}\, U_{t>0}\, (R_1^{\,T} \text{ join } R_3^{\,WO(t)}) \\ &= Ch[A2] / proj_{A2} (R_1^{\,T} \text{ join } R_3^{\,WO}) \end{aligned}$$

where $R_3^{\,WO} = U_{t>0}\ R_3^{\,WO(t)}$. This formula is quite similar to Equation (3) in which $(Ch[\{a_1, a_2\}] / R_3)$ represents all non-allowed traces, while here these traces are represented by $R_3^{\,WO}$. Correspondingly, the only difference in the submodule construction algorithm for the completely defined FSMs that we are interested here and the algorithm described in Section 4.3 is the way the $R_3$ is completed with the introduction of a *fail* state. Here we introduce transitions to the *fail* state from each normal state of the FSM and each input for **all** output values that are different from the original definition of the FSM.

The resulting submodule construction algorithm is essentially identical to the algorithm described in Chapter 6.3.1 of [Kim 97]. However, in our approach we allow for incompletely defined solutions if the behavior for certain states and inputs needs not be defined because such inputs will never happen. We therefore do not introduce explicitly the "{}" state used in [Kim 97]. As an example, we consider the specifications given in Figure 6.5 in [Kim 97] (space limitations do not allow us to give details here). The inputs and outputs labeled x, v, u, z in [Kim 97] correspond to the ports $A_{31}^{\,I}$, $A_{12}^{\,I}$, $A_{12}^{\,O}$, and $A_{31}^{\,O}$, respectively, in Figure 5.1. The interactions at the ports $A_{32}^{\,O}$, and $A_{32}^{\,I}$ do not exist. The completion of the automaton M leads to the following transitions to the *fail* state: from state A under "1/0" and "0/1"; from state B under "0/0" and "1/0". Applying the algorithm described above leads to the same automaton as the one shown in Figure 6.6 in [Kim 97], except for the "{}" state. Since Kim's example does not satisfy the unit-delay constraint, we note that the algorithm described above worksin this case, even though the unit-delay constraint is not satisfied. It is not clear how far the unit-delay assumption may be weakened.

### 5.5 The Case of Interleaving Semantics

In the case of interleaving semantics, there is at each time instant only a real interaction at one of the interfaces, while the other interfaces have the null interaction. In this context, the situation of wrong input is often called "unspecified reception" [Zafi 80].

In this case, there can never be a time instant with wrong input for R1 and wrong output for R3. Therefore the term $(R_1^{\,T})^{WO(t)}$ join $R_3^{\,WO(t)}$ ) in Equation ($3^{IO}$) of the theo-

rem in Section 5.3 is empty and can be dropped. Therefore the equation can be simplied, similarly as in the subsection above, to the form

$$Sol^{(IO)} = Ch[A2] / proj_{A2}\, U_{t>0} (\ (R_1^T \text{ join } R_3^{WO(t)})\ U\ ((R_1^T)^{WO(t)} \text{ join } R_3)\ )$$

$$= Ch[A2] / proj_{A2} (\ (R_1^T \text{ join } R_3^{WO})\ U\ ((R_1^T)^{WO} \text{ join } R_3)\ )$$

If we now consider the case of regular behaviors specified in terms of finite state automata, we come to consider IO-Automata [Lync 89] as finite representation of the regular behaviors. The submodule construction algorithm derived from the above equation is similar to the one for labeled transition systems considered in Section 4.4, except that we have now two classes of non-allowed traces, those giving rise to wrong behavior in respect to $R_3$ and those giving rise to non-expected input to $R_1$. We therefore introduce a *fail* state not only in $R_3$ but also in $R_1$, and all traces in the determinized projected product of $R_1^T$ and $R_3$ that lead to one of these *fail* states must be eliminated. We note this algorithm is essentially identical to the algorithm described in Section 5 of [Dris 99a].

### 5.6 Algorithm for the General Synchronous Case

If we consider regular specifications in the general case of synchronous communication described in Section 5.3, Equation ($3^{IO}$) gives rise to a submodule construction algorithm very similar to the one described in Section 5.5. The main difference is that the synchronous composition operation is used. Again, we have to introduce *fail* states for $R_3$ **and** $R_1$. After forming the product, the projection and the determinization, we have to eliminate all transitions that lead to a state of the determinized specification that contains the *fail* state of either $R_3$ or $R_1$, *and transitions that lead to a state that contains both.* Note that the text in *italics* takes care of the last term "$((R_1^T)^{WO(t)} \text{ join } R_3^{WO(t)})$" in Equation ($3^{IO}$).

## 6 Conclusions

The problem of submodule construction (or equation solving for module composition) has some important applications for the real-time control systems, communication gateway design, and component re-use for system design in general. Several algorithms for solving this problem have been developed based on particular formalisms that were used for defining the dynamic behavior of the desired system and the existing submodule. In this paper, we have shown that this problem and its solution can also be formulated in the context of relational databases.

The main result of this paper is to show that many submodule construction algorithms that have been proposed for different specification paradigms based on finite automata can be derived from this solution of the submodule construction problem within the context of relational databases. In fact, a set-theoretical formulation of this problem has been given in this context and solution formulas have been provided for two cases: (a) when there is no notion of input-output and trace inclusion is taken as conformance relation, and (b) when partial specifications with distinction of input –

output and more complex conformance relations are considered. In both cases, synchronous communication as well as interleaving semantics may be considered.

The solution formula for the case of input-output distinction is new and the corresponding submodule construction algorithm for the corresponding case of general synchronous automata is also new.

In this paper we only consider trace semantics. The considerations of deadlocks and finer conformance relations based on progress or liveness are not considered here. Some references to work in that area are given in [Boch 02a].

**Acknowledgements.** I would like to thank the late Philip Merlin with whom I started to work in the area of submodule construction. I would also like to thank Nina Yevtushenko (Tomsk University, Russia) for many discussions about submodule construction algorithms and the idea that a generalization of the concept could be found for different behavior specification formalisms. I would also like to thank my former colleague Cory Butz for giving a very clear presentation on Bayesian databases which inspired me the database generalization described in Section 3 in this paper. Finally, I would like to thank my former PhD students Z.P. Tao and Jawad Drissi whose work contributed to my understanding of this problem.

## References

[Abad 95] M. Abadi and L. Lamport, Conjoining specifications, ACM Transactions on Programming Languages & Systems, vol.17, no.3, May 1995, pp. 507–34.

[Abit 95] S. Abiteboul, R. Hull and V. Vianu, Foundations of Databases, Add.-Wesley, 1995.

[Boch 80d] G. v. Bochmann and P. M. Merlin, On the construction of communication protocols, ICCC, 1980, pp.371-378, reprinted in "Communication Protocol Modeling", edited by C. Sunshine, Artech House Publ., 1981; russian translation: Problems of Intern. Center for Science and Techn. Information, Moscow, 1981, no. 2, pp. 146–155.

[Boch 01b] G. v. Bochmann, Submodule construction - the inverse of composition, Technical Report, Sept. 2001, University of Ottawa.

[Boch 02a] G. v. Bochmann, *Submodule construction and supervisory control: a generalization*, to appear in Proc. of Int. Conf. on Implementation and Applications of Automata (invited paper), August 2001, Pretoria, South Africa, to be published as Springer Lecture Notes.

[Broy 95] M. Broy, Advanced component interface specification, Proc. TPPP'94, Lecture Notes in CS 907, 1995, pp. 369–392.

[Dris 99a] J. Drissi and G. v. Bochmann, Submodule construction tool, in Proc. Int. Conf. on Computational Intelligence for Modelling, Control and Automation, Vienne, Febr. 1999, (M. Mohammadian, Ed.), IOS Press, pp. 319–324.

[Dris 00] J. Drissi and G. v. Bochmann, Submodule construction for systems of timed I/O automata, submitted for publication, see also J. Drissi, PhD thesis, Univ. of Montreal, March 2000 (in French).

[Hagh 99] E. Haghverdi and H. Ural, Submodule construction from concurrent system specifications, Information and Software Technology, Vo. 41 (1999), pp. 499–506.

[Hoar 85] C. A. R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.

[Kele 94] S. G. H. Kelekar, Synthesis of protocols and protocol converters using the submodule construction approach, Proc. PSTV, XIII, A. Danthine et al (Eds), 1994.

[Kim 72] J.Kim, and M.M.Newborn, The simplification of sequential machines with input restrictions, IRE Trans. on Electronic Computers. December, 1972, pp. 1440–1443.

[Kim 97] T.Kim, T.Villa, R.Brayton, A.Sangiovanni-Vincentelli. Synthesis of FSMs: functional optimization. Kluwer Academic Publishers, 1997.

[Lync 89] N. A. Lynch and M. R. Tuttle, An introduction to input/output automata, CWI Quarterly, 2(3), 1989, pp. 219–246.

[Maie 83] D. Maier, The Theory of Relational Databases, Computer Science Press, Rockville, Maryland, 1983.

[Merl 83] P. Merlin and G. v. Bochmann, On the Construction of Submodule Specifications and Communication Protocols, ACM Trans. on Programming Languages and Systems, Vol. 5, No. 1 (Jan. 1983), pp. 1–25.

[Misr 81] J. Misra and K. M. Chandy, Proofs of networks of processes, IEEE Tr. on SE, Vol. SE-7 (July 1991), pp. 417–426.

[Parr 89] J. Parrow, Submodule Construction as Equation Solving in CCS, Theoretical Computer Science, Vol. 68, 1989.

[Petr 96a] A. Petrenko, N. Yevtushenko, G. v. Bochmann and R. Dssouli, Testing in context: framework and test derivation, Computer Communications Journal, Special issue on Protocol engineering, Vol. 19, 1996, pp.1236–1249.

[Petr 98] A. Petrenko and N. Yevtushenko, Solving asynchronous equations, in Proc. of IFIP FORTE/PSTV'98 Conf., Paris, Chapman-Hall, 1998.

[Qin 91] H. Qin and P. Lewis, Factorisation of finite state machines under strong and observational equivalences, J. of Formal Aspects of Computing, Vol. 3, pp. 284–307, 1991.

[Rama 89] P. J. G. Ramadge and W. M. Wonham, The control of discrete event systems, in Proceedings of the IEEE, Vo. 77, No. 1 (Jan. 1989).

[Yevt 01a] N.Yevtushenko, T.Villa, R.Brayon, A.Petrenko, A.Sangiovanni-Vincentelli. Synthesis by language equation solving (exended abstract), in Proc.of Annual Intern.workshop on Logic Synthesis, 2000, 11–14; complete paper to be published in ICCAD'2001; see also Solving Equations in Logic Synthesis, Technical Report, Tomsk State University,1999, 27 p. (in Russian).

[Zafi 80] P. Zafiropulo, C. H. West, H. Rudin and D. D. Cowan, Towards analyzing and synthesizing protocols, IEEE Tr. Comm. COM-28, 4 (April 1980), pp. 651–660.