Submodule construction – the inverse of composition [*]

Gregor v. Bochmann
School of Information Technology and Engineering (SITE)
University of Ottawa, Canada
(September 6, 2001)

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 ? - This problem has also been called "equation solving", and in the context of supervisory control, it is the problem of designing a suitable controller (second submodule) which controls a given system to be controlled (first submodule). Solutions to this problem have been described in the context of various specification formalisms and various conformance relations.
This paper presents a generalization of this problem and its solution in the context of relational databases, and shows 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 machines 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 and distinguish between input and output interactions. In the sub-case of regular behavior specifications and interleaving semantics, this solution formula gives rise to an algorithm for Input/Output Automata, which is similar to one published recently. The formula also applies to the case of synchronous communication, which was not considered before.

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

[*] This work was partly supported by a research grant from the Natural Sciences and Engineering Research Council of Canada. An overview of the results presented here was presented at the 6-th International Conference on Implementation and Application of Automata, Pretoria, South Africa, on July 23, 2001 (as an invited presentation).

X which is the solution to the equation S1 x X = S. This problem is an analogy of the integer division, which provides the solution to the equation N1 * X = N for integer values N1 and N. In integer arithmetic, there is in general no exact solution to this equation; therefore integer division provides the largest integer which multiplied with N1 is smaller than N. Similarly, in the case of equation solving for machine composition, 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 transition systems (LTS) which communicate with one another by synchronous interactions. This work was later extended to the cases where the behavior of the machines is described in CCS or CSP [Parr 89], by finite state machines (FSM) communicating through message queues [Petr 98] or input/output automata [Dris 99], and to synchronous finite state machines [Qin 91, Kim 97, Yevt 01a].

The applications of this equation-solving method was first considered in the context of the design of communication protocols, where the components S1 and S2 may represent two protocol entities that communicate with one another [Merl 83]. Later it was recognized that this method could also be useful for the design of protocol converters in communication gateways [Kele 94, Tao 97a], and for the selection of test cases for testing a module in a context [Petr 96a]. It is expected that it could also be used in the other application domains where the re-use of components is important. If the specification of the desired system is given together with the specification of a module to be used as one component in the system, then equation solving provides the specification of a new component to be combined with the existing one.

Independently, the same problem was identified in control theory for discrete event systems [Rama 89] as the problem of finding a controller for a given system to be controlled. In this context, the specification S1 of the system to be controlled is given, as well as the specification of certain properties that the overall system, including the controller, should satisfy. If these properties are described by S, and the behavior of the controller is X, then we are looking for the behavior of X such that the equation S1 x X = S is satisfied. Solutions to this problem are described in [Bran 94] using a specification formalism of labeled transition systems where a distinction of input and output is made (interactions of the system to be controlled may be *controllable* (which corresponds to output of the controller) or *uncontrollable* (which correspond to input to the controller). This specification formalism seems to be equivalent to input/output automata (IOA) [Lync 89].

In this paper we show that the above equation solving problem in the different contexts of LTS, communicating finite state machines (synchronous and asynchronous) and IOA are all special cases of a more general problem which can be formulated in the context of relational database theory which is a generalized to allows 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.

These results were obtained after discussions with N. Yevtushenko about the similarity of the formulas that describe the solution of the equation in [Yevt 01a] and [Merl 80]. The generalization described here became apparent after listening to a talk on stochastic relational databases by Cory Butz. In fact, it appears that the solution in the context of relational databases, as described in this paper, can be extended to the case of Bayesian databases.

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. Then we present a general solution formula and its proof. A discussion of related work in the context of other specification formalisms and conformance relations concludes the paper.

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 $D = \cup D_i$ 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 $T: A_r \dashrightarrow D$ with $T(a_i) \in D_i$. An **integrity constraint** is a predicate on such mappings. If the relation R has an integrity constraint C, this means that for each $T \in R$, 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 here 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 $A_1 \cup A_2$, written R_1 join R_2 , with

$T \varepsilon (R_1 \text{ join } R_2)$ 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 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 [xxFSM, Dris 99b].

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

3.1. Some interesting problems (simplest configuration)

In the simple configuration assumed in this subsection, we consider 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 $R_3[\{a_2, a_1\}]$. Their relationship is informally shown in Figure 3.1 .

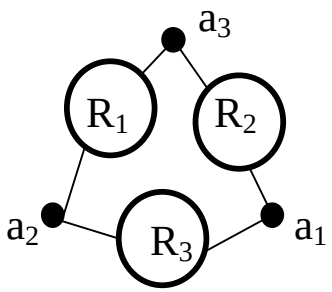

Figure 3.1: Configuration of 3 relations sharing 3 attributes

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

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

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, $R_2 = \Phi$ (empty set), satisfies this equation. However, this case is not very interesting. Therefore we ask the following more interesting questions for the given relations R_1 and R_3 :

Problem (1): Is there a maximal relation R_2 that satisfies the above equation (maximal in the sense of set inclusion; any larger relation is no solution) ?

Problem (2): Could there be more than one maximal solution (clearly not including one another) ?

Problem (3): Is there a solution for the case when the $\subseteq$ operator is replace by equality of by the $\supseteq$ operator ?

3.2. Some solutions

First we note that there is always a single maximal solution. This solution is the set

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

This is true because the operators of set union and intersection obey the distributive law in respect to the projection and join operations, that is, $\text{proj}_{Ax} (R_i \text{ union } R_j) = \text{proj}_{Ax} (R_i) \text{ U } \text{proj}_{Ax} (R_j)$; and similarly for intersection and the join operations.

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):

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

This is the largest solution and all other solutions of Equation (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.

Proof: First we note that ($T_2 \varepsilon Sol^{(3)}$) is equivalent to the statement that there exist no $T \varepsilon Ch[\{a_1, a_2, a_3\}]$ such that **(Equivalence 4)**

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

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

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

Taking any $T' \varepsilon (R_1 \text{ join } Sol^{(3)})$, 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 $T \varepsilon Ch[\{a_1, a_2, a_3\}]$ 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 $T \varepsilon Ch[\{a_1, a_2, a_3\}]$ such that

$proj_{\{a1, a3\}}(T) = T'$ **and** $T_1 =^{def} proj_{\{a2, a3\}}(T) \varepsilon R_1$ **and** $proj_{\{a1, a2\}}(T) \neg\varepsilon R_3$ **(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 ($T' \varepsilon Sol^{(2)}$) implies $proj_{\{a1, a2\}}(\{T'\} \text{ join } \{T_1\}) \subseteq R_3$ while Equation (6) states $proj_{\{a1, a2\}}(\{T'\} \text{ join } \{T_1\}) \neg\varepsilon R_3$. Therefore our assumption must be false. **Q.E.D.**

We note that the smaller solution

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

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 $Sol^{(3*)}$.

3.3. Some simple example

We consider here a very simple example of three relations $R_1[\{a_2, a_3\}]$, $R_2[\{a_1, a_3\}]$, and $R_3[\{a_2, a_1\}]$ as discussed above and shown in Figure 3.1. We assume that the domains of the attributes are as follow: $D_1 = \{n\}$, $D_2 = \{aa, ab, ba, bb\}$ and $D_3 = \{c, d\}$. We assume that R_1 and R_3 contain the tuplets shown in Figure 3.2 below. Then the evaluation of the solution formula Equation (3) leads to some intermediate results and the solution $Sol^{(3)}$, also shown in Figure 3.2.

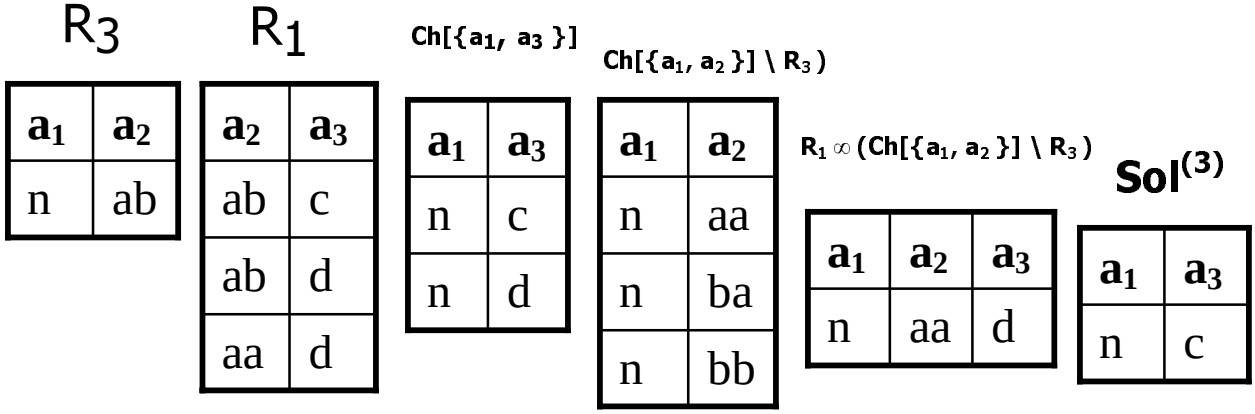

Figure 3.2 : Example of database equation solving (Example 1)

3.4. A more general setting of the problem

In Section 3.2 we assumed that all the three relations have two attributes and that each pair of relations share exactly one attribute. However, we may consider more general situations, such as shown in Figure 3.3. Here we consider different subsets A_1, A_2, A_3 and A_0 of the global set of attributes A. The subsets A_1, A_2, and A_3 correspond to the attributes a_1, a_2, and a_3 considered in Section 3.1, while the subset A_0 is a set of attributes that are shared by all three relations.

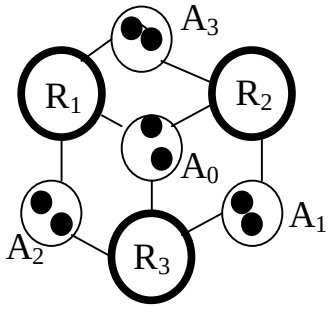

Figure 3.3: Configuration of 3 relations sharing various attributes

The generalization of Equation (1) is then defined as follows. We consider the three relations $R_1[A_2 \cup A_3 \cup A_0]$, $R_2[A_1 \cup A_3 \cup A_0]$, , and $R_3[A_2 \cup A_1 \cup A_0]$. We consider the equation

$$\text{proj}_{(A2 \cup A1 \cup A0)} (R_1 \text{ join } R_2) \subseteq R_3 \qquad \textbf{(Equ. 1')}$$

If the relations R_1 and R_3 are given, the largest relation R_2 that satisfies the above equation is then characterized by the formula

$$Sol^{(3')} = Ch[A_1 \cup A_3 \cup A_0] \ / \ \text{proj}_{(A1 \cup A3 \cup A0)} (R_1 \text{ join } (Ch[A_1 \cup A_2 \cup A_0] \ / \ R_3)) \qquad \textbf{(Equ. 3')}$$

The proof of this equation is similar to the proof of Equation (3).

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*, as shown in Figure 4.1. The ports, shown as lines in Figure 4.1, are the places where the *interactions* between the component in question and the components in its environment take place. Sometimes arrows indicate the direction of the interactions, implying that one component produces the interaction as output while the other component(s) accept it as input. This distinction is further discussed in Section 5.

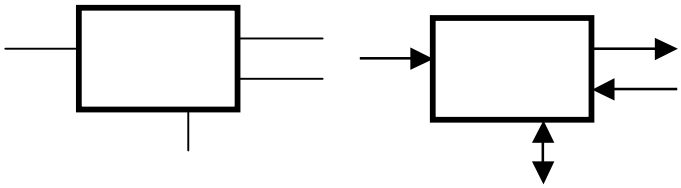

Figure 4.1 : Components and their ports

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 4.2. Their ports are pair-wise interconnected at three interfaces a_1, a_2, and a_3.

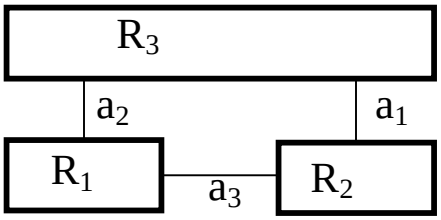

Figure 4.2 : Configuration of 3 components interconnected through 3 interfaces

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 4.2 corresponds to the relationship shown in Figure 3.1. The interfaces a_1, a_2, and a_3 in Figure 4.2 correspond to the three attributes a_1, a_2, and a_3 introduced in Section 2.2, 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.)

As usual, we assume that the possible traces of a module are closed under the prefix relation. Therefore a relation R[A] describing the behavior of a module must also satisfy the following constraint:

Prefix-closure constraint: If $T_x \varepsilon R$ and T_y is such that s_{xi} is a prefix of s_{yi} for all $i \varepsilon A$ (and T_y satisfies the synchrony constraint), then $T_y \varepsilon R$.

As an example we consider two module behaviors R_1 and R_2 which have some similarity with the relations R_1 and R_2 considered in the database example of Section 3.3. These behaviors are described in the form of finite state transition machines in Figure 4.3. The interactions at the interface a_2 are a, b or n, the interactions at a_3 are c, d or n, and the interface a_1 only allows the interaction n. The notation b/n for some state transition means that this transition occurs when at one interface the interaction b occurs and at the other interface the interaction n. For instance, the traces of length 3 defined by the behavior of R_1 are (a/n, n/c. b/n), (a/n, n/d, b/n), and (a/n, n/d, a/n), which are similar, in some sense, to the tuplets in the relation R_1 of the example in Section 3.3.

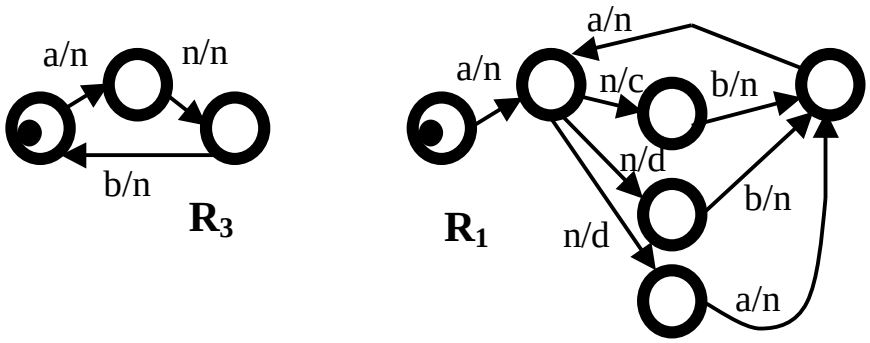

Figure 4.3 : Behavior specifications R_1 and R_2 (Example 2)

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. We note that the algorithm reported in [Yevt 01a] corresponds to Equation (3). Similar work is also described in [Kim 97] and [Qin 91]. In this case, the behavior specification for a module is given in the form of a finite state transition diagram where each transition is labeled by a set of interactions, one for each port of the module, as in the example above.

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 *join* operation corresponds to the composition operator of automata which is of polynomial complexity (see above references for more details). The *substraction* operation is of linear complexity if its two arguments are deterministic. Since the projection operator introduces nondeterminism, one has to include a step to transform the automata into their equivalent deterministic forms. This step is of exponential complexity. Therefore the equation solving algorithm for synchronous finite state machines is 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.

4.4. The case of interleaving rendezvous communication

Under 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). 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 occurred and the interaction v_i which occurred at this 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 simplifies the correspondence with the 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 $T(a_i)[t] \neq$ *null* implies $T(a_j)[t] =$ *null* for all $a_j \in A$ ($j \neq i$)

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 (called stuttering equivalence). One may adopt a normal form representation for such an equivalence class in the form of the execution sequence (in this class) that has no time instance with only null interactions. This execution sequence is trivially isomorphic to the corresponding interaction sequence in the first interleaving model considered above.

We note that we may assume that all relations satisfy the constraint that they are closed under stuttering, that is, $T \in R$ implies that R also contains all other tuplets T' that are stuttering equivalent to T.

4.5. The case of finite labeled transition systems

The interleaving rendezvous communication is adopted for labeled transition systems (LTS) (voir e.g. [Hoare 85]). 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 $a_i \neq a_j$), and we introduce the overall set of interactions $I = U_{(ai \in A)} I_i$. Then a class of stuttering equivalent interleaving traces (as described in Section 4.4) correspond 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 and the operations *projection*, *join* and *substraction* over interleaving traces can be represented by finite operations over the corresponding LTS representations. The situation is similar as in the case of synchronous finite state machines, discussed in Section 4.3, because of the nondeterminism introduced by the project operator, the substraction operation becomes of exponential complexity. The projection operation corresponds to replacing the interaction labels of transitions that correspond to ports that are not included in the projected set by a spontaneous transition label (sometimes written "i"). The join operation is the standard LTS composition operation, and the determination and substraction operations can be found in standard text books of automata theory.

As an example, we may consider the behavior specifications given in Figure 4.3. If we interpret the interaction "n" as the null interaction, then the behaviors R_1 and R_3 satisfy the interleaving constraint described above and can be interpreted as labeled transition

systems. Their traces can be characterized by the regular expressions " (a . b)* " and " (a . (c . b + d . b + d . a))* ", respectively. If we execute the algorithm implied by Equation (3) we obtain the solution behavior for R_2 which can be characterized by "c*". This solution is similar to the solution for the database example discussed in Section 3.3.

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 elemenbt 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:

(a) Valid trace: A tuplet (trace) T is valid in respect to a relation (specification) R if $T \varepsilon R$.

(b) Equivalence of specifications: Two relations (specifications) are equivalent iff they are equal (i.e. they contain the same traces).

(c) Trace inclusion: An implementation conforms to a specification R iff all possible traces of the implementation are valid in respect to R.

In the case that we distinguish between inputs and outputs, we can still use the above definition of a valid trace, but the conformance relations between implementation and specification, or between different specifications are more complex, as described below.

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 [Adab 94, 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:

(d) 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.

(e) A trace specification (relation) $R[A_R^I \mid A_R^O]$ satisfies the UDC iff for any $T \varepsilon R$ the following holds:

$$\text{next}(T, R) = \text{proj}_{A_R^I}(\text{next}(T, R)) \quad \text{join} \quad \text{proj}_{A_R^O}(\text{next}(T, R))$$

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 ε R), we may consider the longest valid prefix of T; there must exist a time instant t > 0 such that $T^{|t-1}$ ε R and $T^{@t}$ ¬ε next($T^{|t-1}$, R) (we use the notation where ¬ε 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:

(i) Wrong output: We say that T has wrong output at time t, written T ε $R^{WO(t)}$, iff $T^{|t-1}$ ε R and proj $_{A_R^O}$ $T^{@t}$ ¬ε proj $_{A_R^O}$ next($T^{|t-1}$, R).

(ii) Wrong input: We say that T has wrong input at time t, written T ε $R^{WI(t)}$, iff $T^{|t-1}$ ε R and proj $_{A_R^I}$ $T^{@t}$ ¬ ε proj $_{A_R^I}$ next($T^{|t-1}$, 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 sat_G R, iff for all t > 0 the following holds: $T^{|t-1}$ ε R implies T ¬ ε $R^{WO(t)}$.

(2) A trace T over A **satisfies the hypotheses of R**, written T sat_H R, iff for all t > 0 the following holds: $T^{|t-1}$ ε R implies T ¬ ε $R^{WI(t)}$.

(3) A trace T over A **satisfies the specification R**, written T sat R, iff (T sat_H R) implies (T 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 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:

(1) the guarantees of R_3 are satisfied, and

(2) the hypotheses of R_1 are satisfied.

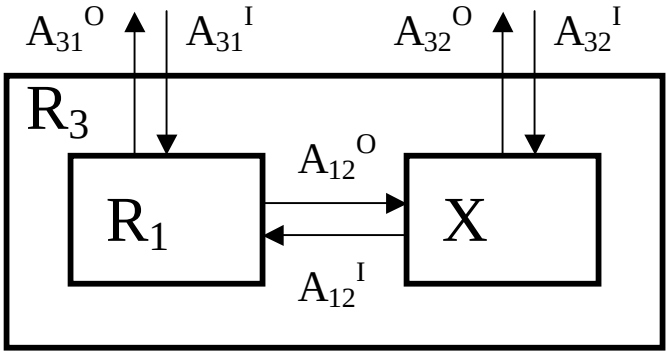

Figure 5.1 : Composition of components R_1 and X with input/output interactions

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[A1] which satisfies Equation (1^{IO}) below:

$proj_{A3}$ (R_1 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] / $proj_{A2}$ $U_{t>0}$ (**(Equ. 3^{IO})**

(R_1 join $R_3^{WO(t)}$) U ($R_1^{WI(t)}$ join R_3) U ($R_1^{WI(t)}$ join $R_3^{WO(t)}$))

To prove the correctness of this solution, we proceed in a similar manner as for the proof of Equation (3) in Section 3.2. We consider an arbitrary trace T produced by the composition of R_1 with the solution $Sol^{(IO)}$, and show that it satisfies the requirement of the problem. We have to prove that the following property holds for any trace T over A:

[$proj_{A1}$(T) sat_H R_1 implies $proj_{A1}$(T) sat_G R_1] **and**
T ε $Sol^{(IO)}$ **and** [$proj_{A3}$(T) sat_H R_3] **(Equ. 7)**
implies [$proj_{A3}$(T) sat_G R_3]

This property can be proven by induction on the length of T. As the basis for the induction, we consider the trace of zero length for which the property is satisfied.

It is easily seen that the induction step is essentially equivalent to showing the following property. This property states that if the prefix of length (t-1) of T has the property that its projections onto A1, A2, and A3 belong to the specifications R_1,

$Sol^{(IO)}$ and R_3, respectively, then T will have at the next time instant (a) correct output in respect to R_3 and (b) correct input in respect to R_1. Formally:

$(proj_{A1}(T^{|t-1}) \varepsilon R_1)$ **and** $(proj_{A2}(T) \varepsilon Sol^{(IO)})$ **and** $(proj_{A3}(T^{|t-1}) \varepsilon R_3)$
and $(proj_{A3}(T^{|t}) \neg\varepsilon R_3^{WI(t)})$ **implies** **(Equ. 8)**
$(proj_{A3}(T^{|t}) \neg\varepsilon R_3^{WO(t)})$ **and** $(proj_{A1}(T^{|t}) \neg\varepsilon R_1^{WI(t)})$

The consequence of Equation (8), together with the antecedent of Equation (7), implies that $(proj_{A1}(T^{|t}) \varepsilon R_1)$ **and** $(proj_{A3}(T^{|t}) \varepsilon R_3)$ are true, thus validating the antecedent of Equation (8) for the next induction step (i.e. for time t).

In order to show that Equation (8) holds, we use the fact that $(proj_{A2}(T) \varepsilon Sol^{(IO)})$. According to the definition of $Sol^{(IO)}$, this is equivalent to saying that there exists no T' over A such that $(proj_{A2}(T') = proj_{A2}(T))$ and for some $t > 0$ any of the following cases is true:

(a) $proj_{A1}(T') \varepsilon R_1$ and $proj_{A3}(T') \varepsilon R_3^{WO(t)}$
(b) $proj_{A1}(T') \varepsilon R_1^{WI(t)}$ and $proj_{A3}(T') \varepsilon R_3$
(c) $proj_{A1}(T') \varepsilon R_1^{WI(t)}$ and $proj_{A3}(T') \varepsilon R_3^{WO(t)}$

Therefore, the properties (a), (b) and (c) must also be false for T. We note that falseness of (a) and (c) implies that $proj_{A3}(T)$ not $\varepsilon R_3^{WO(t)}$, and the falseness of (b) and (c) implies that $proj_{A1}(T)$ not $\varepsilon R_1^{WI(t)}$ (which proves the consequence in Equation (8)).
The proof that $Sol^{(IO)}$ is the largest solution is similar as the corresponding proof in Section 3.2.

5.4. 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 has also been 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 $(R_1^{WI(t)}$ join $R_3^{WO(t)})$ in the formula for $Sol^{(IO)}$ in the Theorem of Section 5.3 is empty and can be dropped.

We note that the algorithm described in Section 5 of [Dris 99b] corresponds to the formula of the above theorem for the case of regular behavior specifications in the form of Input/Output Automata.

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 can also be formulated in the context of relational databases. The solution to the problem is given

in the form of a set-theoretical formula which defines the largest relation that is a solution of the equation.

Whether this solution is useful for practical applications in the context of relational databases is not clear. However, we have shown here that the formulation of this problem in the context of relational databases is a generalization of several of the earlier approaches to submodule construction, in particular in the context of synchronous finite state machines [Kim 97, Yevt 01a], Labelled Transition Systems (LTS) [Merl 83], and Input/Output Automata (IOA) [Dris 99c], and it was explained in [Dris 99b] that the case of IOA is a generalization of the case of communicating finite state machines with queued interactions [Petr 98]. In the case of regular behavior specifications in the form of finite transition machines, the set-theoretical solution formula of the database context can be used to derive solution algorithms based on the finite representations of the module behaviors, which correspond to those described in the literature.

The solution formula for the case of synchronous communication with the distinction of input and output (implying a module specification paradigm with hypothesis and guarantees, as discussed in section 5) has not been described before, as far as I know. It can be used to derive an algorithm to solve the submodule construction problem for synchronous communicating machines when a distinction between input and output is made. This context has not yet been considered for the problem of submodule construction.

We believe that these solution formulas can also be used to derive submodule construction algorithms for specification formalism that consider finer conformance relations than simple trace semantics (as considered in this paper). Examples of existing algorithms of this class are described in [This 95] for considering liveness properties and in [Bran 94, Male95, Dris 00] for considering hard real-time properties. Some other work [Parr 89] was done in the context of the specification formalism CSP [Hoare 85] and observational equivalence for which it is known that no solution algorithm exists because the problem is undecidable.

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, Addison-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.

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

[Bran 94] B. A. Brandin and W. M. Wonham, Supervisory Control of Timed Discrete-Event Systems, IEEE Tran. on Automatic Control, Vol.39, No.2, Feb. 1994.

[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 99b] J. Drissi and G. v. Bochmann, Submodule construction for systems of I/O automata, submitted for publication.

[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, University of Montreal, March 2000 (in French).

[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 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.

[Male 95] O. Maler, A. Pnueli and J. Sifakis, On the synthesis of discrete controllers for timed systems, STACS 95, Annual Symp. on Theoretical Aspects of Computer Science, Berlin, 1995, Springer Verlag, pp. 229-242.

[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, Journal 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).

[Tao 97a] Z. Tao, G. v. Bochmann and R. Dssouli, A formal method for synthesizing optimized protocol converters and its application to mobile data networks, Mobile Networks & Applications, vol.2, no.3, 1997, pp.259-69. Publisher: Baltzer; ACM Press, Netherlands.

[Tao 95d] Z. P. Tao, G. v. Bochmann and R. Dssouli, A model and an algorithm of subsystem construction, in proceedings of the Eighth International Conference on parallel and distributed computing systems, Sept. 21-23, 1995 Orlando, Florida, USA, pp.619-622.

[This 95] J. G. Thistle, On control of systems modelled as deterministic Rabin automata, Discrete Event Dynamic Systems: Theory and Applications, Vol. 5, No. 4 (Sept. 1995), pp. 357-381.

[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 Snthesis, 2000, 11-14; complete paper to be published in ICCAD'2001; see also Solving Equations in Logic Synthesis, Technical Report, Tomsk State University, Томск, 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.