Using First-Order Logic to Reason about Submodule Construction[1]

Gregor v. Bochmann

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

A shorter version of this paper was included in the proceedings of the IFIP international conference on Formal Techniques for Distributed Systems (FMOODS/FORTE), 2009

Abstract. We consider the following problem: For a system consisting of two components, the behavior of one component is known as well as the desired global behavior. What should be the behavior of the second component such that the behavior of the composition of the two conforms to the desired behavior ? - This problem has been called "submodule construction" or "equation solving"; and in the context of supervisory control, it is the problem of designing a suitable controller (second component) which controls a given system to be controlled (first component). Solutions to this problem have been described in the context of various specification formalisms and various conformance relations. This paper presents a new formulation of this problem and its solution in first-order logic. It is also shown how the solutions for submodule construction in various specification formalisms can be derived from the solution in logic. The simple proof of correctness for the logic solution is then used to justify the particular forms of solutions in the different specification formalisms, such as (a) synchronous rendezvous at several interfaces, and (b) interleaved rendezvous (labeled transition systems).

1. Introduction

In automata theory, the notion of constructing a product machine S from two given finite state machines M_A and M_B, written $M = M_A \times M_B$, is a well-known concept (see Figure 1(a)). 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, called “equation solving” or “submodule construction”: Given the composed system M and one of the components M_A, what should be the behavior of the second component M_B such that the composition of these two components M_A and M_B will exhibit a behavior equal to M. That is, we are looking for the value of X

[1] *This work was partly supported by a research grant from the Natural Sciences and Engineering Research Council of Canada.*

which is the solution to the equation M_A x X = M (see Figure 1(b)). This problem is an analogy of 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 M_A satisfies some conformance relation in respect to M. In the simplest case, this conformance relation is trace inclusion.

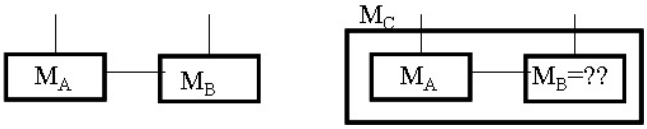

Fig. 1. (a) two communicating components; (b) submodule construction problem

A first paper of 1980 [1] (see also [2]) 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 interleaved rendezvous interactions (see also [3] for a more formal treatment). This work was later extended to the cases where the behavior of the machines is described in CSP [4] (with behavioral equivalence as conformance relation) [5], by finite state machines (FSM) communicating through message queues [6], by input/output automata (IOA) [7, 8, 9] ([7] considers bisimulation as conformance relation), and by synchronous finite state machines [10, 11]. The case of extended state machine models including state variables and assertions about input and output parameters has also been studied [12]. The problem has also been formulated for databases using relational algebra [13].

One application of this submodule construction method was considered in the context of the design of communication protocols, where the components M_A and M_B may represent two protocol entities that communicate with one another [2]. Later it was recognized that this method could also be useful for the design of protocol converters in communication gateways [15, 16], and for the selection of test cases for testing a module in a context [17].

Independently, the same problem was identified in control theory for discrete event systems [18] as the problem of finding a controller for a given system to be controlled. In this context, the specification M_A 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 M, and the behavior of the controller is X, then we are looking for the behavior of X such that the equation M_A x X = M is satisfied. Solutions to this problem are described in [19] 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 generated by the controller) or *uncontrollable* (which corresponds to input to the controller). This specification formalism seems to be equivalent to input/output automata (IOA) [20].

In some private discussions, Nina Yevtushenko pointed out that the formulas that describe the solutions of the equations for synchronous and interleaving automata, as described in [11] and [1], respectively, have a quite similar structure. Later, when

listening to a talk on stochastic relational databases by Cory Butz, I noticed that the same kind of equation solving problem can be formulated in the context of relational databases, and that the solution has again a quite similar structure. I have argued [13, 9] that the problem of equation solving in relational databases is a generalization of the problem for synchronous and interleaving automata, and that also a solution for IOA can be derived from this more general problem. During the discussion after the presentation of my paper [9] at the FORTE conference, the question was raised whether this equation solving problem could also be formulated within the context of first-order logic.

The purpose of this paper is to show that, in fact, the equation solving (or submodule construction) problem can be formulated in logic. It turns out that (a) a solution with a structure similar to the solutions mentioned above exists, and (b) a proof of the correctness of this solution is quite simple, apparently much simpler than the existing proofs of correctness for the solutions in the contexts mentioned above. We show in this paper how the solutions for submodule construction in different contexts can be derived from the general solution in the logic context. The proof of correctness from the logic context can therefore be used to justify the particular forms of solutions in the contexts of different specification formalisms. We consider in this paper the context of communicating system components using (a) synchronous rendezvous at several interfaces, or (b) interleaved rendezvous (that is, labeled transition systems). Other contexts are considered in [32], such as synchronous (I/O) automata with complete or partial behavior specifications, interleaving IOA with complete or partial behavior specifications, and finite state machines with queued communication, as well as relational algebra for databases. These contexts include much of the previous work mentioned above and also some not so common modeling approaches.

The paper is structured as follows: The next section presents the problem of equation solving in the general context of first-order logic. The main concepts and equations are established which are then referenced in the subsequent sections. In Section 3, the submodule construction problem is introduced in the context of modular system design where the overall system is composed out of several components and the behavior of one of the components is to be found. A modeling framework for synchronous communication between all the components is introduced. Section 4 shows how this modeling framework can be used to model interleaving semantics, as used by labeled transition systems (LTS). Section 5 presents the conclusions.

2. Equation solving in the logic context

2.1. The logic context

We use in this section first-order logic with typed variables. We consider a universe with three variables X_A, X_B, and X_C that may take values from three domains D_A , D_B and D_C , respectively. These domains may be infinite. Therefore, the set of possible

value assignments to the variables is $U = D_A \times D_B \times D_C$. We write x_A, x_B, and x_C for possible values of the variables X_A, X_B, and X_C , respectively.

We are interested in relationships between values of different variables. For instance, we may consider a relation $R \subset D_A \times D_B$ which is a subset of pairs $< x_A, x_B >$ of values of the variables X_A and X_B . We also use predicates to characterize sets. For instance, the relation R may be characterized by a predicate $C(x_A, x_B)$ which is true exactly for those pairs $< x_A, x_B >$ that are in R.

2.2. The equation solving problem

In the following, we are interested in three relations $R_A \subset D_B \times D_C$, $R_B \subset D_A \times D_C$ and $R_C \subset D_A \times D_B$. We write $C_A(x_B, x_C)$, $C_B(x_A, x_C)$, and $C_C(x_A, x_B)$ for their respective characterizing predicates. We consider the following proposition which relates these three relations:

$$\forall < x_A, x_B, x_C > \in U : <x_B, x_C> \in R_A \ \wedge \ <x_A, x_C> \in R_B \ \Rightarrow \ <x_A, x_B> \in R_C \quad (1^{Rel})$$

This proposition may be equivalently rewritten in terms of the predicates as follows:

$$\forall < x_A, x_B, x_C > \in U : \ C_A(x_B, x_C) \wedge C_B(x_A, x_C) \ \Rightarrow \ C_C(x_A, x_B) \qquad (1^{Pred})$$

The problem of equation solving is the following: We assume that R_A and R_C are given. What are the properties of relation R_B that ensure that proposition (1) is satisfied? – We would like to find a maximal solution R_B^{max} to this problem, that is, R_B^{max} together with R_A and R_C would satisfy (1), but any larger $R_B' \supset R_B^{max}$ would not satisfy this proposition.

2.3. The maximal solution

Starting from (1^{Pred}), it is easy to see that the following predicate characterizes the maximal solution:

$$C_B^{max}(x_A, x_C) \ = \ \forall \ x_B \in D_B : \ C_A(x_B, x_C) \ \Rightarrow \ C_C(x_A, x_B) \qquad (2)$$

The right side of this definition can be equivalently transformed in several steps as follows:

$$\forall \ x_B \in D_B : \ \neg C_A(x_B, x_C) \ \vee \ C_C(x_A, x_B)$$
$$\forall \ x_B \in D_B : \ \neg \ (\ C_A(x_B, x_C) \ \wedge \ \neg C_C(x_A, x_B) \ )$$
$$\neg \exists \ x_B \in D_B : \ C_A(x_B, x_C) \ \wedge \ \neg C_C(x_A, x_B)$$

which leads to the following equivalent expression for the maximal solution:

$$C_B^{max} \ (x_A, x_C) \ = \ \neg \exists \ x_B \in D_B : \ C_A(x_B, x_C) \ \wedge \ \neg C_C(x_A, x_B) \qquad (3)$$

2.4. The realized subset of R_C

We note that in general not all pairs $<x_A, x_B> \in R_C$ could be "realized" by R_A and R_B^{max}.

Definition: We say that a pair $<x_A, x_B> \in R_C$ is **realizable** by R_A and R_B if there exist a value $x_C \in D_C$ such that $<x_B, x_C> \in R_A$ and $<x_A, x_C> \in R_B$.

We call the subset of R_C that is realisable by R_A and R_B^{max} the maximally realisable subset of R_C (or "product"), written R_C^{prod}. We therefore have

$$<x_A, x_B> \in R_C^{prod} \text{ iff } \exists\ x_C \in D_C : <x_B, x_C> \in R_A \wedge <x_A, x_C> \in R_B^{max} \qquad (4)$$

2.5. The reduced maximal solution

We consider the relation $R_B^{incompatible}$ characterized by the following predicate:

$$C_B^{incompatible}(x_A, x_C) = \neg\exists\ x_B \in D_B :\ C_A(x_B, x_C) \wedge C_C(x_A, x_B)$$

Lemma: There is no $<x_A, x_B> \in R_C$ that is realizable by R_A and $R_B^{incompatible}$.

Proof: Let us assume that there is a pair $<x_A, x_B> \in R_C$ that is realizable by R_A and $R_B^{incompatible}$. According to the definition of "realizable", this implies that there is a $x_C \in D_C$ such that $<x_B, x_C> \in R_A$ and $<x_A, x_C> \in R_B^{incompatible}$. Now, the definition of $R_B^{incompatible}$ implies that there is no $x'_B \in D_B$ such that $C_A(x'_B, x_C) \wedge C_C(x_A, x'_B)$. However, this is a contradiction, since x_B satisfies this condition for x'_B.

We conclude from the lemma above that those pairs $<x_A, x_C>$ of R_B^{max} that are in $R_B^{incompatible}$ do not contribute to the realization of R_C^{prod}. We therefore may eliminate from the solution R_B^{max} all pairs in $R_B^{incompatible}$ and still obtain the same set R_C^{prod} of realizable pairs $<x_A, x_B>$. We call this the **reduced maximal solution** to the equation solving problem. It is characterized by the following predicate:

$$C_B^{red}(x_A, x_C) = (\ \exists\ x_B \in D_B :\ C_A(x_B, x_C) \wedge C_C(x_A, x_B)\ ) \wedge (\ \neg\exists\ x_B \in D_B :\ C_A(x_B, x_C) \wedge \neg C_C(x_A, x_B)\ ) \qquad (5)$$

2.6. Example

$D_A = \{a_1, a_2\}$; $D_B = \{b_1, b_2, b_3\}$; $D_C = \{c_1, c_2, c_3, c_4\}$;
$R_A = \{<b_1, c_1>, <b_2, c_2>, <b_1, c_3>, <b_2, c_3>, <b_3, c_3>\}$;
$R_C = \{<a_1, b_1>, <a_2, b_2>, <a_1, b_3>\}$;
The relation corresponding to the predicate $\neg C_C(x_A, x_B)$ is the complement of R_C in respect to the set of all tuplets in $D_A \times D_B$, written $\neg R_C$. The set of all tuplets in $D_A \times D_B$ is also sometimes called the "chaos" over $D_A \times D_B$, written $Chaos_{A\times B}$. We have $Chaos_{A\times B} = \{<a_1, b_1>, <a_2, b_2>, <a_1, b_3>, <a_1, b_2>, <a_2, b_1>, <a_2, b_3>\}$, where the last three tuplets are in $\neg R_C$.

Using Formula (3), we obtain

$$R_B^{max} = Chaos_{A\times C} \setminus \{<a_1, c_2>, <a_2, c_1>, <a_1, c_3>, <a_2, c_3>\}$$
$$= \{<a_1, c_1>, <a_2, c_2>, <a_1, c_4>, <a_2, c_4>\}$$

where "\" is the set subtraction operator.

We have $R_C^{prod} = \{<a_1, b_1>, <a_2, b_2>\}$ which is a subset of R_C .

We note that the tuplets $<a_1, c_3>$ and $<a_2, c_3>$ are in $R_B^{incompatible}$, and the reduced maximal solution is $R_B^{red} = \{<a_1, c_1>, <a_2, c_2>\}$.

3. Submodule construction for synchronous systems

3.1. Modeling systems consisting of several components

State machines are often used as models for reactive systems that interact with their environment. Often one considers a system model which is the composition of several state machines. Therefore a state machine is normally a component within a system, it interacts with other components of the system and possibly also with the environment of the system; or the state machine represents the interactions of the whole system with its environment.

A system component has one or more interfaces. An interface is a location where interactions with the environment of the component take place. Each interface i is associated with a domain I_i ; the elements of I_i are the possible interactions that may take place at that interface during a given time unit. We write $x_i^{(t)}$ for the interaction that takes place at interface i at time unit t. Clearly, $x_i^{(t)} \in I_i$ for all t. We write x_i for a sequence of interactions at interface i over a certain time period. We write I_i^* for the set of all sequences that can be formed by concatenating interactions from the domain I_i . We have $x_i \in I_i^*$.

We assume trace semantics for specification of the dynamic behaviour of a system, that is, the dynamic behavior of a system M is defined in terms of the set of possible execution histories that could occur during the execution of the component. For a system with n interfaces i (i = 1, …, n), an execution history consists of a tuplet $< x_1, x_2, \ldots x_n >$ where x_i (i = 1, …, n) is the sequence of interactions that occurred at interface i during the execution history. We therefore assume that the specification S of the dynamic behavior of M is given in the form of a (normally infinite) set of such tuplets. As in Section 2, instead of talking about the set S of tuplets, one may also talk about the predicate C that characterizes this set.

Let us assume that the system consists of a certain number of components (sub-systems) C_j (j = 1, …m), each connected to a certain number of interfaces. For instance C_1 may be connected to interfaces i = 1, 3 and 6; and its behaviour predicate, therefore, will be of the form $C_1(x_1, x_3, x_6)$. Let us assume that the system has performed synchronous interactions over t time units and the execution history $eh = < x_1, x_2, \ldots x_n >$ has been observed. Since all components were involved in this execution, the behaviour predicates of all components must be true for the interface interaction sequences x_i in *eh*.

We now ask the question: What could be the interactions $x_i^{(t+1)}$ at the interfaces (i = 1, …, n) during the next time unit (t+1). These interactions must satisfy the condition that the extended interaction sequences including that time unit, $x_i' = x_i$ *concatenated-with* $x_i^{(t+1)}$, should also satisfy the behaviour predicates of all the

components. This means that all components must be ready to engage in the interactions $x_i^{(t+1)}$ at those interfaces to which they are connected. In other words, there is a kind of global rendezvous between all the components to agree on a set of interactions $x_i^{(t+1)}$ at the interfaces that are agreeable to all component. This also means that any resulting execution history will satisfy all component behaviour predicates; in other words, the behaviour predicate of a composition of several components is the conjunction of the behaviour predicates of all participating components.

We note that the definition of this kind of synchronous rendezvous occurring simultaneously at several interfaces is a concept that would be difficult to be implemented in a distributed context. It is not clear whether this is a concept of practical importance, however, we think that it exhibits a theoretical simplicity that makes it interesting. Other more practical communication paradigms are discussed in the subsequent sections.

Besides composition, there is another important operation for describing the behaviour of a system consisting of several components. This is the hiding of an interface that is not visible from a certain perspective. Let us consider a system configuration consisting of several components and n interfaces i (i = 1, …, n). We assume that the dynamic behaviour of the system is characterized by the predicate $C(x_1, x_2, \ldots x_n)$. When one of the interfaces (say i) is hidden, we obtain a visible behaviour which only involves the non-hidden interfaces. We use the notation "$\text{hide}^{(syn)}_i\,(C(x_1, x_2, \ldots x_n)$" to represent the predicate of this behaviour. As discussed in [23], this predicate has the following form:

$$< x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n > \in \text{hide}^{(syn)}_i\,(C(x_1, x_2, \ldots x_n))$$
$$\text{iff } \exists\, x_i \in I_i^* : < x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n > \in C(x_1, x_2, \ldots x_n)$$

3.2. Submodule construction

We now consider a system configuration containing two components M_A and M_B as shown in Figure 2(a). Since the sequences at the three interfaces are constrained by the behaviour of the two components, we have the following predicate that characterizes the set of all possible execution histories of this system:

$$\forall < x_A, x_B, x_C > \in U : \; C_A(x_B, x_C) \wedge C_B(x_A, x_C)$$

where $U = I_A^* \times I_B^* \times I_C^*$ is the universal set of execution sequences for a system architecture as shown in Figure 2(a).

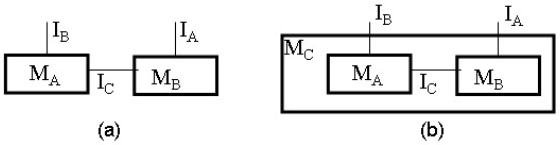

Fig. 2. Two components M_A and M_B; (b) also showing the desired overall behavior M_C

Let us now assume that the system consisting of the composition of the two components M_A and M_B is supposed to behave like a system M_C characterized by the

predicate $C_C(x_A, x_B)$, as shown in Figure 2(b). Then we have the following requirement:

$$\forall < x_A, x_B, x_C > \in U : \; C_A(x_B, x_C) \wedge C_B(x_A, x_C) \; = \; C_C(x_A, x_B)$$

If we suppose that the behavior defined by $C_C(x_A, x_B)$ represents a safety requirement, that is, all execution histories generated by the two components M_A and M_B must satisfy $C_C(x_A, x_B)$, then we have the requirement:

$$\forall < x_A, x_B, x_C > \in U : \; C_A(x_B, x_C) \wedge C_B(x_A, x_C) \; \Rightarrow \; C_C(x_A, x_B) \qquad (1^{syn})$$

This formula is identical to Equation (1^{Pred}) if we assume that the domains D_i of Section 2 are sets of interaction sequences, namely $D_i = I_i^*$ (for i = A, B and C).

The equation solving problem introduced in Section 2 becomes, in the context of interacting state machines, the following "submodule construction problem". We assume a system structure as shown in Figure 2(b). If the specification of M_A is given in the form of $C_A(x_B, x_C)$, as well as the safety requirement $C_C(x_A, x_B)$ for the overall system, what is the most relaxed requirement for the dynamic behaviour of machine M_B ?

Since the proposition (1^{syn}) is identical to (1^{Pred}), we can use Formula (3) to obtain the most general behaviour of M_B that satisfies (1^{syn}). Using the hiding operator discussed in Section 3.1, we may rewrite Formula (3) as follows:

$$C_B^{max}(x_A, x_C) \; = \; \neg \, hide^{(syn)}_B \, (\, C_A(x_B, x_C) \; \wedge \; \neg C_C(x_A, x_B) \;)$$

We note that the negation can also be expressed by the complement in respect to the corresponding chaos behaviour, as in the Formula (3^{RelAlg}) of Section 2.7. This leads to the following:

$$C_B^{max}(x_A, x_C) = I_A^* \times I_C^* \setminus hide^{(syn)}_B \, ((C_A(x_B, x_C) \; \wedge (I_A^* \times I_B^* \setminus C_C(x_A, x_B))) \qquad (3^{syn})$$

It is important to note that the operators used in Formula (3^{syn}) can be evaluated algorithmically if the infinite sets of sequences defined by $C_A(x_B, x_C)$ and $C_C(x_A, x_B)$ are regular sets, that is, are defined by a state machine with a finite number of states. The number of states of the machine representing the composition of two machines is smaller or equal to the product of the number of states of the two machines. The complement is straightforward to calculate in the case of deterministic state machines and can be realized by complementing the accepting and non-accepting states. Unfortunately, the hiding operator introduces in general non-determinism and the resulting non-deterministic behaviour description must be transformed into a deterministic description which is a problem of exponential complexity (see for instance [24]). We conclude that the complexity of the sub-module construction problem for regular state machines is in general exponential.

The considerations concerning the realizability of all the sequences defined by $C_C(x_A, x_B)$ and the reduced maximal solution C_B^{red} are identical to what was discussed in Section 2. We note that one would normally be interested in the reduced maximal solution for the component M_B which is given by the following formula:

$$C_B^{red}(x_A, x_C) \; = \; hide^{(syn)}_B \, (C_A(x_B, x_C) \; \wedge \; C_C(x_A, x_B) \,) \\ \setminus \; hide^{(syn)}_B \, (\, (C_A(x_B, x_C) \; \wedge \; (I_A^* \times I_B^* \setminus C_C(x_A, x_B) \,) \,) \qquad (5^{syn})$$

3.3. Examples

We present here two examples which are very similar to the example of Section 2.6. If we take the variable values a_i, b_j, and c_k of Section 2.6 as the possible interactions at the interfaces I_A, I_B and I_C, respectively, and impose no constraints on the order in which these interactions may take place, we obtain state diagrams for M_A and M_C with a single state having a transition for each tuplet in the respective relation R_A or R_C defined in Section 2.6. For example, for M_C we obtain the state diagram of Figure 3(a), and similarly for M_A . This example is isomorph to the example of Section 2.6 and yields as reduced maximal solution a machine that allows the synchronous transitions $<a_1, c_1>$ and $<a_2, c_2>$ in an arbitrary order.

To make this example a bit more interesting, we consider now that the behaviour of M_C is given by the state diagram of Figure 3(b) which means that synchronous interaction tuplets $<a_1, b_1>$ and $<a_2, b_2>$ must alternate and one or more $<a_1, b_3>$ may be inserted after a $<a_1, b_1>$. In this case we obtain the reduced maximal solution shown in Figure 3(c).

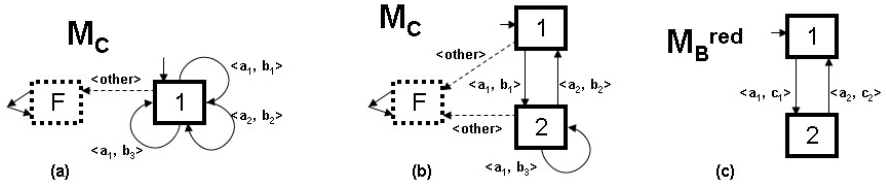

Fig. 3. (a) behavior of a synchronous machine M_C corresponding to the example of Section 2.6; (b) modified behavior of machine M_C; (c) reduced maximal solution for the modified behavior of M_C

4. Submodule construction for labeled transition systems

4.1. Modeling interleaving semantics

In this modeling framework, we also have rendezvous interactions at interfaces, but interleaving semantics is assumed, which means that at most one interaction (on a single interface) may occur during each time unit. We use in the following the same modelling framework used for synchronous machines, but introduce the following changes:

- We allow an interface to have the value *null* during a given time unit, which means that no interaction takes place at this interface during this time unit.
- In a system of several components with n interfaces, a possible execution history $< x_1, x_2, \ldots x_n >$ must satisfy the following constraint, called **interleaving constraint**:

  $IC(x_1, x_2, \ldots x_n)$ = for all t : $x_i^{(t)} \in I_i$ implies $x_j^{(t)} = null$ for all $j \neq i$.

Any execution history $h = < x_1, x_2, \ldots x_n >$ that satisfies the interleaving constraint defines a linear (time) order for the (non-null) interactions at the interfaces. We write seq(h) for this sequence and call it the execution sequence corresponding to

h. For execution sequences over n interfaces, as above, we have seq(h) $\in (I_1 \cup I_2 \ldots \cup I_n)^*$. Normally, the semantics of labelled transition systems is described in terms of these (finite and infinite) execution sequences and the possibilities of blocking after finite sequences. We will continue using the model of separate interaction sequences x_i at the different interfaces, as introduced for synchronous communication; we thus obtain a uniform framework for treating systems with both types of communication, synchronous and interleaving.

Definition (Equivalence of execution histories): Since only the execution sequences count for the semantics of labelled transition systems, we say that two execution sequences h_1 and h_2 are equivalent, written $h_1 \cong h_2$, if they define the same execution sequence, that is, $seq(h_1) = seq(h_2)$.

This corresponds to the so-called stuttering equivalence between execution sequences that contain at certain time units null-interactions at all interfaces. Clearly, we assume that any predicate defining the behaviour of a given system component has the same value for equivalent execution histories; the value should only depend on the corresponding execution sequence.

The notion of equivalence between execution histories leads to a slightly modified definition of the hiding operator as follows:

$< x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n > \in \text{hide}^{(LTS)}{}_i\,(C(x_1, x_2, \ldots x_n))$

iff $IC(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$

$\wedge \; \exists < x_1', \ldots, x_{i-1}', x_i', x_{i+1}', \ldots, x_n' > :$ ($IC(x_1', \ldots, x_{i-1}', x_i', x_{i+1}', \ldots, x_n')$

$\wedge \; < x_1', \ldots, x_{i-1}', x_{i+1}', \ldots, x_n' > \cong < x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n >$

$\wedge \; < x_1', \ldots, x_{i-1}', x_i', x_{i+1}', \ldots, x_n' > \in C(x_1, x_2, \ldots x_n)$)

4.2. Submodule construction

Due to the interleaving constraints and the equivalence between execution histories, we have the following modified equations. Equation (1^{syn}) becomes:

$\forall < x_A, x_B, x_C > \in U : IC(x_A, x_B, x_C) \wedge C_A(x_B, x_C) \wedge C_B(x_A, x_C) \Rightarrow C_C(x_A, x_B)$ (1^{LTS})

Equation (2) becomes:

$C_B{}^{max}(x_A, x_C) = IC(x_A, x_C) \wedge \forall < x_A', x_B', x_C' > \in U :$

$IC(x_A', x_B', x_C') \wedge <x_A', x_C'> \cong <x_A, x_C> \wedge C_A(x_B', x_C') \Rightarrow C_C(x_A', x_B')$ (2^{LTS})

This definition of $C_B{}^{max}$ says that an execution history at the interfaces I_A and I_C is an allowed behavior for component M_B if for all global execution histories $< x_A', x_B', x_C' >$ that have an equivalent behavior for M_B, the satisfaction of C_A leads to the satisfaction of C_A . This modification to Equation (2) is introduced because the specification of the behavior for M_B can only restrain the possible execution sequences of the component, but has no impact on which of the equivalent execution histories would be realized in collaboration with the other system components and the environment.

Using a similar demonstration as for Equation (3) in Section 2, it is easy to see that Equation (2^{LTS}) is equivalent to

$$C_B^{max}(x_A, x_C) = IC(x_A, x_C) \wedge \neg\exists < x_A', x_B', x_C' > \in U : IC(x_A', x_B', x_C') \wedge <x_A', x_C'> \cong <x_A, x_C> \wedge C_A(x_B', x_C') \wedge \neg C_C(x_A', x_B')$$

Using the definition of the hiding operator above, this can be rewritten in the form

$$C_B^{max}(x_A, x_C) = IC(x_A, x_C) \wedge \neg\, hide^{(as)}_B (C_A(x_B, x_C) \wedge \neg C_C(x_A, x_B)) \quad (3^{LTS})$$

Using a similar demonstration as for Equation (5) in Section 2, we can show that the reduced maximal solution is given by the formula

$$C_B^{red}(x_A, x_C) = hide^{(LTS)}_B (C_A(x_B, x_C) \wedge C_C(x_A, x_B) \wedge \neg\, hide^{(LTS)}_B (C_A(x_B, x_C) \wedge \neg C_C(x_A, x_B)) \quad (5^{LTS})$$

This solution was presented (using a different notation) in [1], which was the first paper on submodule construction to our knowledge. We note that this formula is the same as (5^{syn}), except that a different hiding operator is used.

Like in the case of synchronous machines, these solutions may be evaluated algorithmically when the specifications of C_A and C_C are given in the form of regular languages (finite state machines).

4.3. Example

An example is shown in Figure 4. State diagrams representing the behaviour of M_A and M_C are shown (solid boxes) in Figure 4(a) and (b). The state diagram for M_C also shows the **Fail** (F) state (a non-accepting state) and as well as the non-allowed transitions (dotted arrows). The behaviour of ($C_A(x_B, x_C) \wedge C_C(x_A, x_B)$) is shown in Figure 4(c); it represents the composition of the two state machines M_A and M_C . The diagram is incomplete, since transitions from a composed state where M_C is in the Fail state are not shown.

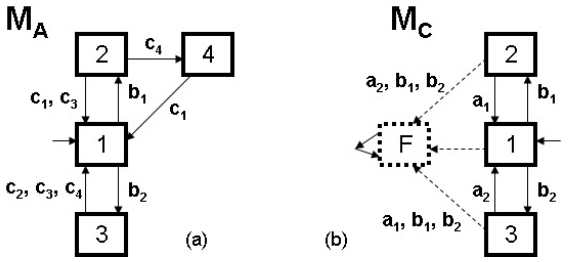

Fig. 4. (a, b) State diagrams representing the behavior of M_A and M_C

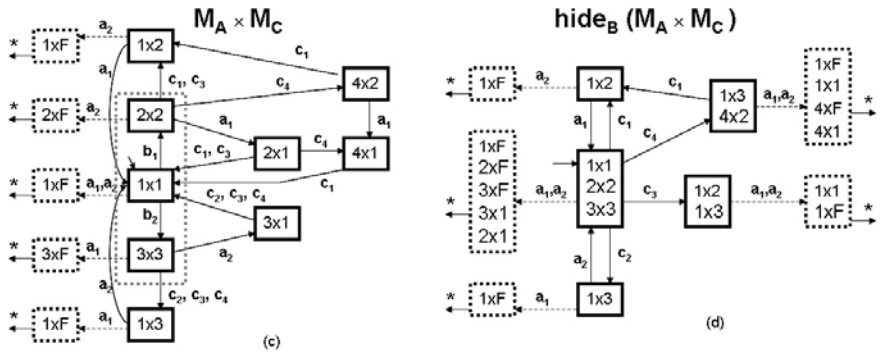

Fig. 4. (c) Behavior of the composition of M_A and M_C; (d) the same after hiding interactions b_1 and b_2 and determination

Figure 4(d) shows the state diagram obtained from Figure 4(c) after the following two steps: (a) hiding the interactions b_1 and b_1 at the interface I_B , and (b) transforming the resulting non-deterministic machine into an equivalent deterministic one. Each state of the deterministic machine represent a subset of the states of the original non-deterministic machine, namely those that could be reached through transitions including the hidden ones (see for instance [24] for details). Since Equation (5^{LTS}) only considers execution sequences that can be generated jointly by M_A and M_C and that can not reach the Fail state of M_A, we do not need to explore the transitions for the resulting deterministic machine from any state that includes in its subset a state pair for which M_C is in the Fail state. We conclude that Equation (5^{LTS}) results in the behaviour for M_B as shown by the full arrows of Figure 4(d).

4.4. Avoidance of deadlocks

The solution equations discussed above do not consider the possibility of deadlocks since their derivation is based on trace semantics where the meaning of the specification of a component is the set of possible execution histories or execution sequences. In the context of submodule construction, it is common practice to avoid deadlocks by pruning those transitions in the obtained solution behaviour for M_B that may lead to a deadlock. There are two steps in this pruning process:

- If the obtained specification of M_B (considered alone) contains a deadlock state (such as the state {<1×2>, <1×3>} in Figure 4(d)) then all transitions leading to this state should be pruned. This may in turn introduce other deadlock states for which the same kind of pruning should be performed, etc. (In the example of Figure 4(d), the transition labelled c_4 should be eliminated).
- Further deadlocks may be detected when the joint behaviour of M_A and M_B is evaluated (namely C_C^{prod} as defined in Equation (4)). Again, transitions in M_B that lead to a joint deadlock state should be pruned. As under Step (1) above, this may be a recursive process. Finally, a joint behaviour without deadlock is obtained; we may call this the "maximal non-blocking behaviour" for M_C , or $C_C^{non\text{-}block}$. In

certain cases, this behaviour may contain no transition, that is, the system blocks; this means that the given behaviour of M_A is in some sense incompatible with the required system behaviour C_C .

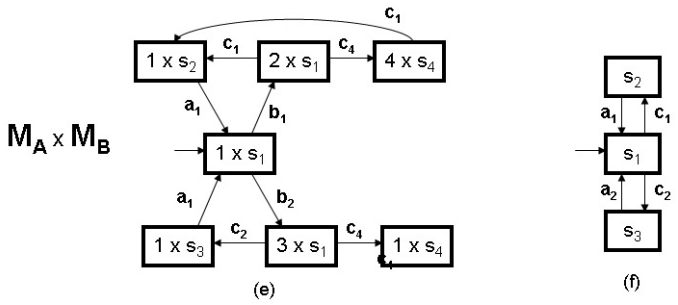

Fig. 4. (e) behavior (containing a potential deadlock) obtained by the composition of M_A and the M_B obtained in Step 1 above; (f) final specification of the behavior of M_B (avoiding the deadlock)

In this example, the joint behaviour of M_A and the behaviour of M_B after Step (1) above is shown in Figure 4(e). (Note: the states s_i are the states shown in Figure 4d: s_1 = {<1,1>, <2,2>, <3,4>}, s_2 = {<1,2>}, s_3 = {<1,4>}, s_4 = {<1,3>, <4,2>}). It contains a deadlock state, which may be eliminated by pruning the transition c_4 in the behaviour of M_B and the subsequent state s_4 = {<1×3>, <4×2>}. This leads to the behaviour for M_B shown in Figure 4(f). In this example, the whole required behaviour C_C is realized, that is, $C_C^{\text{non-block}} = C_C$.

7. Conclusions

The problem of submodule construction (or sometimes called equation solving) has some important applications for real-time control systems, communication gateway design, testing of embedded components, and component re-use for system design in general. Several algorithms for solving this problem have been developed based on different formalisms that are 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 a more general setting using first-order logic. It turns out that solutions to this problem in logic are quite simple. We show in this paper that these solutions (and their proof of correctness) can be mapped into the different specification formalisms considered in the earlier work. Therefore this paper provides, in a sense, new proofs of correctness for the solutions of the submodule construction problem described in earlier work.

The different specification formalisms considered are system components using synchronized rendezvous interactions on several interfaces, or rendezvous with interleaving semantics (e.g. labeled transition systems). Input/output interactions and state machines with queued message passing are considered in a different paper [32]. It is important to note that in the case of regular behaviour specifications in the form

of finite state machines, the solutions to the submodule construction problem can derived by an algorithm which is, in general, of exponential complexity.

We consider in this paper trace semantics, that is, the behaviour of the system, or of a component, is characterized by the set of possible execution histories. This is adequate for safety properties, but ignores issues of liveness, progress, absence of deadlocks and fairness (with the exception of Section 4.4 where deadlock avoidance is discussed). We believe that the issues of hard real-time properties (see for instance [19, 27, 28]) could also addressed with the approach presented in this paper. However, we are not sure whether it could be helpful for dealing with liveness and progress properties (as for instance discussed in [29, 30, 28, 31]).

It is to be noted that the complexity of the algorithms for deriving the submodule construction solution depends on the specification formalism used. As mentioned above, it is exponential for finite state behavior descriptions, however, it is polynomial if the interactions at the interface I_B are not hidden; on the other hand, it has been shown to be undecidable for behavior specifications in CSP [5].

References

[1] 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.

[2] 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.

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

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

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

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

[7] 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.

[8] 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.

[9] G. v. Bochmann, Submodule construction for specifications with input assumptions and output guarantees, in Proc. FORTE'02 (22st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems), Chapman&Hall, 2002, pp.

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

[11] 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 in Conference on Computer-Aided Design (ICCAD '01), 2001, pp. 103; see also Solving Equations in Logic Synthesis, Technical Report, Tomsk State University, Томск, 1999, 27 p. (in Russian).

[12] B. Daou and G. v. Bochmann, Submodule construction for extended state machine models, Proc. IFIP Intern. Conf. on Formal Techniques for Networked and Distributed Systems - FORTE 2005, Taiwan, 2005, Springer LNCS 3731, 2005, pp. 396-410.

[13] G. v. Bochmann, Submodule construction and supervisory control: a generalization, in Proc. of Int. Conf. on Implementation and Applications of Automata, Aug. 2001 (invited paper), Springer Lecture Notes, 2002.
[14] J.Kim, and M.M.Newborn, The simplification of sequential machines with input restrictions, IRE Trans. on Electronic Computers. December, 1972, pp. 1440-1443.
[15] S. G. H. Kelekar, Synthesis of protocols and protocol converters using the submodule construction approach, Proc. PSTV, XIII, A. Danthine et al (Eds), 1994.
[16] 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.
[17] 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.
[18] 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).
[19] 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.
[20] N. A. Lynch and M. R. Tuttle, An introduction to input/output automata, CWI Quarterly, 2(3), 1989, pp. 219-246.
[21] D. Maier, The Theory of Relational Databases, Computer Science Press, Rockville, Maryland, 1983.
[22] S. Abiteboul, R. Hull and V. Vianu, Foundations of Databases, Addison-Wesley, 1995.
[23] M. Abadi and L. Lamport, Conjoining specifications, ACM Transactions on Programming Languages & Systems, vol.17, no.3, May 1995, pp. 507-34.
[24] A. V. Aho, R. Sethi and J. D. Ullman, Compilers, Principles, Techniques and Tools, Addison
Wesley, 1986.
[25] J. Misra and K. M. Chandy, Proofs of networks of processes, IEEE Tr. on SE, Vol. SE-7 (July 1991), pp. 417-426.
[26] M. Broy, Advanced component interface specification, Proc. TPPP'94, Lecture Notes in CS 907, 1995, pp. 369-392.
[27] 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.
[28] 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).
[29] 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.
[30] 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.
[31] S. Buffalov, K. El-Fakih, N. Yevtushenko and G. v. Bochmann, Progressive solutions to a parallel automata equation, Proc. FORTE Conf. (IFIP), Sept. 2003, Berlin, LNCS 2767, Springer Verlag, pp. 367-382.
[32] G.v. Bochmann, Submodule construction – the inverse of composition, in preparation.