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

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". 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 which is the solution to the equation $M_A \times X = M$ (see Figure 1(b)). Actually, since equality often cannot be realized, we are looking for the most general machine X which composed with M_A satisfies some conformance relation in respect to M. In this paper we consider trace inclusion as conformance relation.

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

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 LTS (communicating by interleaved rendezvous). This work was later extended to the cases where the behavior of the machines is described by CSP, FSMs with queues, IOAs and synchronous FSMs. The problem has also been formulated for databases using relational algebra (see [3] for pertinent references). The main applications of this work are in the design of communication protocols, the construction of protocol converters for communication gateways, the selection of test cases of testing a module in a context, and for finding a controller for discrete event control systems [4].

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. In this paper we give an overview of the cases of (a) synchronous rendezvous at several interfaces, and (b) interleaved rendezvous (that is, labeled transition systems). A more detailed discussion, including examples, can be found in [3]. Other contexts are considered in [5], 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.

2. Equation solving in 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.

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 now consider the following proposition:

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

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.

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:

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

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

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

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:

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

3. Submodule construction for synchronous systems and LTS

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. Because of space limitations, this sections is much condensed. More details can be found in [3].

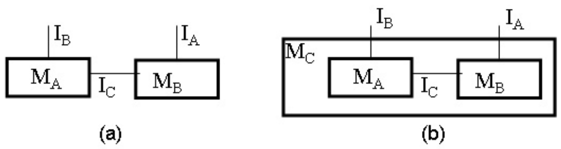

Fig. 1. (a) two communicating components; (b) submodule construction problem

A system component has one or more interfaces 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. 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 the 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 $< x_1, x_2, \dots x_n >$ 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 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.

3.1. Submodule construction for synchronous systems

For synchronous systems, there is an interaction at each interface during each global time unit. Therefore, for a system as shown in Figure 1(b), the formula

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

states that the traces of the composition of machines M_A and M_B are included in the traces of M_C . In order to compare this formula with what has been discussed previously in the literature, we have to introduce the hiding operator. When one of the interfaces (say i) is hidden, we obtain a visible behaviour which only involves the non-hidden interfaces. For a behavior C of a machine with n interfaces, we use the notation "$\text{hide}^{(syn)}_i (C(x_1, x_2, \ldots x_n)$" to represent the predicate of the behaviour when interface i is hidden. As discussed by Abadi and Lamport, this predicate has the following form:

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

We note that (1^{syn}) has the form of (1^{Pred}) and we can follow the derivations of Sections 2.3 through 2.5. Using the above formula for hiding, we can rewrite Equation (5) of Section 2 for the reduced maximal solution as follows:

$$\begin{aligned} C_B^{red} (x_A, x_C) = \; & \text{hide}^{(syn)}_B (C_A(x_B, x_C) \wedge C_C(x_A, x_B)) \\ & \setminus \text{hide}^{(syn)}_B ((C_A(x_B, x_C) \wedge (I_A^* \times I_B^* \setminus C_C(x_A, x_B))) \quad (5^{syn}) \end{aligned}$$

3.2. Submodule construction for 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 as 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**:

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

We say that two execution histories are equivalent if they exhibit the same sequence of non-null interactions. This leads to the following formula of interface hiding:

$$\begin{aligned} & < x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n > \in \text{hide}^{(LTS)}_i (C(x_1, x_2, \ldots x_n)) \\ & \quad \text{iff } IC(x_1, \ldots , x_{i-1}, x_{i+1}, \ldots, x_n) \\ & \quad \wedge \; \exists < x_1', \ldots , x_{i-1}', x_i', x_{i+1}', \ldots, x_n' > : (IC(x_1', \ldots , x_{i-1}', x_i', x_{i+1}', \ldots, x_n') \\ & \qquad \wedge \; < x_1', \ldots , x_{i-1}', x_{i+1}', \ldots, x_n' > \cong < x_1, \ldots , x_{i-1}, x_{i+1}, \ldots, x_n > \\ & \qquad \wedge \; < x_1', \ldots , x_{i-1}', x_i', x_{i+1}', \ldots, x_n' > \in C(x_1, x_2, \ldots x_n)) \end{aligned}$$

Because of the interleaving constraint, the formulas (1^{syn}) and (2) become:

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

$$C_B^{max}(x_A, x_C) = IC(x_A, x_C) \wedge \forall < x_A', x_B', x_C' > \in U :$$
$$IC(x_A', x_B', x_C') \wedge <x_A', x_C'> \cong <x_A, x_C> \wedge C_A(x_B', x_C') \Rightarrow C_C(x_A', x_B') \quad (2^{LTS})$$

And the reduced maximal solution becomes

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

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.

4. Conclusions

We have shown in this paper that the problem of submodule construction can be formulated in a general setting using first-order logic. It turns out that solutions to this problem in logic are quite simple, and they can be mapped (together with their proof of correctness) into the different specification formalisms considered in earlier work. Therefore this paper provides, in a sense, new proofs of correctness for the solutions of the submodule construction problem described earlier.

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.

It is to be noted that the complexity of the algorithms for constructing the missing submodule depends on the specification formalism used. For state machines the complexity is polynomial if the interactions at the interface I_C in Figure 1(b) are visible by M_C, however, if they are hidden, as we assume in this paper, the algorithms become exponential, because the hiding introduces non-determinism and the algorithm to find a deterministic automaton equivalent to a non-deterministic one is exponential. The problem becomes undecidable for behavior specifications in CSP.

References

For a complete list of references, see [3].

[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] Extended version of this paper, see http://www.site.uottawa.ca/~bochmann/dsrg/PublicDocuments/Publications/Boch09a.pdf

[4] 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).

[5] G.v. Bochmann, Submodule construction – the inverse of composition, submitted.