ELSEVIER

Information and Software Technology 41 (1999) 499–506

INFORMATION AND SOFTWARE TECHNOLOGY

Submodule construction from concurrent system specifications ☆

E. Haghverdi, H. Ural*

School of Information Technology and Engineering, MacDonald Hall, 150 Louis Pasteur, University of Ottawa, Ottawa, Ontario KIN 6N5, Canada

Received 1 June 1998; received in revised form 17 December 1998; accepted 17 February 1999

Abstract

The submodule construction problem (SCP) as stated and formulated by Merlin and Bochmann [P. Merlin, G.V. Bochmann, On the construction of submodule specification and communication protocols, ACM Trans. Prog. Lang. Sys., 5(1) (1983) 1–25] is considered: given the specification of a system (module) and that of its $n-1$ submodules, determine the specification of the nth submodule that together with the given $n-1$ submodules will satisfy the given system specification. We recast SCP in a formal setting and proceed to present and prove the correctness of an algorithm for the solution of SCP where submodules are prefix-closed finite state machines. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Concurrent systems; Stepwise refinement; Submodule construction problem

1. Introduction

The problem of designing concurrent systems for distributed applications has been an active area during the last decade. The complexity of designing such systems has led the designers to stepwise refinement techniques where the specification of a given system is decomposed into interacting modules which in turn are decomposed into submodules until a satisfactory level of decomposition of the system functionality is achieved. The following problem, referred to as submodule construction problem (SCP), arises in this context: given the specification of a system (module) and that of its $n-1$ submodules, determine the specification of the nth submodule that together with the given $n-1$ submodules will satisfy the given system specification. A particular instance of this problem can be given for communication protocols. In this case the specifications for the communication services to be provided by the protocol and by the underlying layer are given. The specification of one of the protocol entities is also given and the task is to derive the specification of the other protocol entity [8,10].

SCP is formulated and treated by Merlin and Bochmann [4]. They give a formula which defines the specification of the unknown submodule in the general case where submodule specifications are given in terms of sets of possible execution sequences. They find the most general specification possible for the unknown submodule. The treatment and presentation of [4] is to a large extent informal. This obscures the main ideas presented in the article and makes a mathematical and formal analysis of the material presented therein very difficult.

In this article we recast the material presented in [4] in a more formal and mathematically more precise way. We next consider the case where the submodules are given by finite state machines (FSM) and present an algorithm for SCP. The algorithm is then proven to be correct.

Beside treating the material in [4] in a more formal way, our work extends theirs along the following lines: (1) we present and prove correct an algorithm for the solution of SCP in the case of FSM modules, whereas in [4] no explicit algorithm is given for the solution of SCP. (2) Although not explicitly mentioned in [4], the examples considered there involve FSMs with all states as accepting states. Our proposed algorithm, however, handles the general case where each module is considered to be a prefix-closed FSM (i.e. it accepts prefix-closed languages) with arbitrary accepting states (cf. Example 1). Hence, the examples treated in [4] are special cases of SCP that can be solved by our algorithm (cf. Example 3). (3). The proposed algorithm produces a prefix-closed FSM, however the method in [4] does not yield a prefix-closed module in general.

The rest of the article is organized as follows: in Section 2 we recast the material presented in [4] and formulate the main result presented there in Proposition 1 and Theorem 1.

☆ This work is supported in part by the Natural Sciences and Engineering Research Council of Canada under grant number STR0149338.

* Corresponding author. Tel.: + 1-613-562-5800; fax: + 1-613-562-5185.

E-mail address: ural@site.uottawa.ca (H. Ural)

0950-5849/99/$ - see front matter © 1999 Elsevier Science B.V. All rights reserved.
PII: S0950-5849(99)00014-2

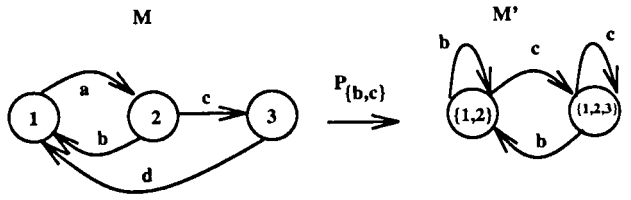

Fig. 1. Projection example.

In Section 3 we present our algorithm for SCP and prove it correct. This is followed by some examples in Section 4. Finally in Section 5 we discuss the remaining problems, the related work and conclude with proposed directions for future work.

2. Submodule construction problem

Throughout this section, M_i is used to denote a module or a submodule, V_i is used to denote the set of atomic actions that are executed by M_i, and S_i is used to denote the specification of M_i as the set of all possible execution sequences of M_i. Let M_0 be a module which is to be decomposed into submodules M_1 and M_2 and let specifications of M_0 and M_1 be given. Let V_0 and V_1 denote the sets of atomic actions and let S_0 and S_1 denote the sets of all possible execution sequences of M_0 and M_1 respectively. Clearly $S_0 \subseteq V_0{}^*$ and $S_1 \subseteq V_1{}^*$. $\bar{S}_0$, the complement of S_0 is taken with respect to $V_0{}^*$ i.e. $\bar{S}_0 = V_0{}^* - S_0$. The submodule construction problem (SCP) can be formulated as follows:

Given module M_0 and submodule M_1 and the sets V_0, S_0, V_1 and S_1, find a submodule M_2 with $S_2 \subseteq V_2{}^*, V_2 = (V_0 - V_1) \cup (V_1 - V_0)$ such that:

$$P_{V_0}(S_1 \times S_2) \subseteq S_0$$

where the product ' $\times$ ' is defined as: $S_j \times S_k = \{s \in (V_j \cup V_k)^* | P_{V_j}(s) \in S_j \wedge P_{V_k}(s) \in S_k\}$ where $S_j \subseteq V_j{}^*$ and $S_k \subseteq V_k{}^*$ and projection $P_V : U^* \rightarrow V^*$ with U, V two sets of actions and $V \subseteq U$, is defined as:

$$P_V(\varepsilon) = \varepsilon$$

$$P_V(a{\cdot}s) = \begin{cases} a{\cdot}P_V(s), & \text{if } a \in V; \\ P_V(s) & \text{if } a \notin V. \end{cases}$$

Note that V_2 consists of those actions that are not both in V_0 and V_1. That is, the actions on which M_0 and M_1 synchronize (i.e. the actions appearing with the same name both in V_0 and V_1) are not in the action set V_2 of M_2.

The above formulation of SCP considers the case where a given module M_0 is decomposed into two submodules, M_1 and M_2, i.e. $n = 2$. As stated in Section 1, this formulation adheres to the stepwise refinement approach where the specification of a module is decomposed into interacting submodules which in turn are decomposed into their submodules until a satisfactory level of decomposition is achieved. That is, when $n > 2$, M_0 stands for the given module, M_1 stands for the composition of the given $n - 1$ submodules, and M_2 stands for the nth submodule. Below we give the proof of the proposition given in [4] for completeness.

Proposition 1 (Merlin and Bochmann [4]). *Suppose $V_2 \subseteq V_0 \cup V_1$, then the sequences s_2 in $P_{V_2}(S_0 \times S_1)$ are exactly those sequences over V_2 for which*

$$P_{V_0}(S_1 \times \{s_2\}) \cap S_0 \neq \emptyset.$$

Proof. Let $V = V_0 \cup V_1, s_2 \in P_{V_2}(S_0 \times S_1) \Leftrightarrow \exists s \in S_0 \times S_1, s \in V^*$ such that $P_{V_2}(s) = s_2 \Leftrightarrow P_{V_0}(s) \in S_0$ and $P_{V_1}(s) \in S_1$ and $P_{V_2}(s) = s_2 \Leftrightarrow \exists s_0 \in S_0, s_0 \in V_0{}^*$ such that $P_{V_0}(s) = s_0$ and $P_{V_1}(s) \in S_1$ and $P_{V_2}(s) = s_2 \Leftrightarrow \exists s_0 \in S_0$ such that $P_{V_0}(s) = s_0$ and $s \in (S_1 \times \{s_2\}) \Leftrightarrow \exists s_0 \in S_0$ such that $s_0 \in P_{V_0}(S_1 \times \{s_2\})$. But $s_0 \in S_0$, so $s_0 \in P_{V_0}(S_1 \times \{s_2\}) \cap S_0$, hence $P_{V_0}(S_1 \times \{s_2\}) \cap S_0 \neq \emptyset$ where $\emptyset$ denotes the empty set. □

Corollary 1. The sequences s_2 in $P_{V_2}(\bar{S}_0 \times S_1)$ are exactly those sequences on V_2 for which $P_{V_0}(\bar{S}_1 \times \{s_2\}) \cap \bar{S}_0 \neq \emptyset$.

Below we formalize the *Statement about the Construction of Submodule Specifications* in [4] as Theorem 1 and provide a formal proof.

Theorem 1. *Given M_0, V_0, S_0, M_1, V_1, S_1, and $V_2 = (V_0 - V_1) \cup (V_1 - V_0), S_2 = P_{V_2}(S_0 \times S_1) - P_{V_2}(\bar{S}_0 \times S_1)$ is the maximal set satisfying the following; $P_{V_0}(S_1 \times S_2) \subseteq S_0$.*

Proof. For $s_2 \in S_2$, we have:

$s_2 \in P_{V_2}(S_0 \times S_1)$, hence by Proposition 1 $P_{V_0}(S_1 \times \{s_2\}) \cap S_0 \neq \emptyset$ and $s_2 \notin P_{V_2}(\bar{S}_0 \times S_1)$. Hence by Corollary 1 $P_{V_0}(S_1 \times \{s_2\}) \cap \bar{S}_0 = \emptyset$, therefore $P_{V_0}(S_1 \times \{s_2\}) \subseteq S_0$. To see this, let $s \in P_{V_0}(S_1 \times \{s_2\}) - S_0$, then $s \in P_{V_0}(S_1 \times \{s_2\}) \cap \bar{S}_0$, a contradiction. Note that $s_2 \in S_2$ was chosen arbitrarily and therefore we have $P_{V_0}(S_1 \times S_2) \subseteq S_0$ as desired.

For maximality of S_2, let $S_2' \subseteq V_2{}^*$ and $S_2 \subset S_2'$. Let $s \in S_2'$ and $P_{V_0}(S_1 \times \{s\}) \neq \emptyset$. We have $P_{V_0}(S_1 \times \{s\}) \subseteq S_0$, hence $P_{V_0}(S_1 \times \{s\}) \cap S_0 \neq \emptyset$ and $P_{V_0}(S_1 \times \{s\}) \cap \bar{S}_0 = \emptyset$. By Proposition 1 and Corollary 1, $s \in S_2 and S_2' = S_2$. □

Next, we give the proposed algorithm for the solution of SCP and prove its correctness.

3. Proposed algorithm

Definition 1. A finite state machine (FSM) M is a quintuple $(Q, \Sigma, \delta, q0, F)$, where Q is the finite set of states, Σ a finite set of actions, $\delta : Q \times \Sigma \rightarrow Q$ is the transition partial

function. $q0 \in Q$ is the initial state, and F finite set of final (accepting) states. $Q - F$ is the set of nonaccepting states and δ can be extended to strings on Σ in the standard way and is denoted here by δ^*. $L(M) = \{s \in \Sigma^* | \delta^*(q0, s) \in F\}$ denotes the language accepted by M. For $q \in Q, \Sigma(q)$ denotes the set of actions for which δ is defined at q, i.e., $\Sigma(q) = \{\sigma \in \Sigma | \delta(q, \sigma)$ is defined$\}$.

We use M, M', and M_i to denote FSMs.

Definition 2. Given an FSM $M = (Q, \Sigma, \delta, q0, F)$ with $F \neq Q$, the *complement* of M is an FSM $\bar{M} = (\bar{Q}, \bar{\Sigma}, \bar{\delta}, \overline{q0}, \bar{F})$ defined as: $\bar{Q} = Q, \bar{\Sigma} = \Sigma, \bar{\delta} = \delta, \overline{q0} = q0$ and $\bar{F} = Q - F$.

The case where $F = Q$ involves the addition of an extra state called the *dead state* with the necessary transitions. A dead state is a nonaccepting state. There is a transition from each state to the dead state on each action in the action set for which there is no transition defined for the action in that state. There is a transition from the dead state to itself for each action in the action set (for details see [3]).

Definition 3. Given an FSM $M = (Q, \Sigma, \delta, q0, F)$, let $L(M)$ be the language accepted by M. Pre($L(M)$) is the largest prefix-closed subset of $L(M)$. Hence Pre($L(M)$) $= L(M)$ iff M is prefix-closed.

With (sub) modules as prefix-closed FSMs, we have the following correspondence of the notation used in this section to the notation used in Section 2: in FSM M_i, Σ_i corresponds to V_i and $L(M_i)$ corresponds to Pre(S_i).

Definition 4. Given $M_1 = (Q_1, \Sigma_1, \delta_1, q0_1, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q0_2, F_2)$, their *synchronous product* $M = M_1 \times M_2 = (Q, \Sigma, \delta, q0, F)$ is defined as follows: $Q = \{(q_1, q_2) \in Q_1 \times Q_2 | \exists s \in \Sigma^*, \delta^*((q0_1, q0_2), s) = (q_1, q_2)\}$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $q0 = (q0_1, q0_2)$, $F = F_1 \times F_2$, and

$$\delta((q_1, q_2), \sigma) = \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)), & \text{if } \sigma \in \Sigma_1(q_1) \cap \Sigma_2(q_2); \\ (\delta_1(q_1, \sigma), q_2), & \text{if } \sigma \in \Sigma_1(q_1) - \Sigma_2; \\ (q_1, \delta_2(q_2, \sigma)), & \text{if } \sigma \in \Sigma_2(q_2) - \Sigma_1; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

The synchronous product defines the interaction between FSMs by requiring the actions with the same name in two FSMs be executed jointly. Such an action cannot be executed by a single FSM if the other one is not ready to execute the same action.

Every action of M_1 or M_2 occurring in any $s \in L(M_1 \times M_2)$ is possible iff M_1 or M_2 participates in that action, respectively. Hence $L(M_1 \times M_2) = \{s \in (\Sigma_1 \cup \Sigma_2)^* | P_{\Sigma_1}(s) \in L(M_1)$ and $P_{\Sigma_2}(s) \in L(M_2)\}$. Examples of the synchronous product are given in Section 4 as part of the examples of the application of the proposed method.

Definition 5. Given an FSM $M = (Q, \Sigma, \delta, q0, F)$ and a set $\Delta \subseteq \Sigma$, the Δ-Closure of $q \in Q$, denoted as $\mathrm{Cls}_\Delta(q)$, is defined as:

$$\mathrm{Cls}_\Delta(q) = \{q' \in Q | \exists s \in \Delta^*, \delta^*(q, s) = q'\}$$

Note that $q \in \mathrm{Cls}_\Delta(q)$ since $q = \delta^*(q, \varepsilon)$ and $\varepsilon \in \Delta^*$.

Definition 6. Given $M = (Q, \Sigma, \delta, q0, F)$ and $\Sigma' \subseteq \Sigma$, let $\Delta = \Sigma - \Sigma'$. We define $P_{\Sigma'}(M) = (Q', \Sigma', \delta', p0, F')$ as follows: $Q' = \{p \in (2^Q - \{\emptyset\}) | \exists s \in \Sigma'^*, \delta'^*(p0, s) = p\}$, $p0 = \mathrm{Cls}_\Delta(q0)$, $\delta' : Q' \times \Sigma' \to Q'$, $\delta'(p, \sigma) = \cup_{g \in p} \mathrm{Cls}_\Delta(\delta(q, \sigma))$. We set $\mathrm{Cls}_\Delta(\delta(q, \sigma)) = \emptyset$ whenever $\delta(q, \sigma)$ is undefined and we declare $\delta'(p, \sigma)$ undefined whenever $\delta'(p, \sigma) = \emptyset$. Finally $F' = \{p \in Q' | p \cap F \neq \emptyset\}$.

Fig. 1 presents the projection of a given FSM M over $\{b,c\}$ as $P_{\{b,c\}}(M)$.

Lemma 1. *Let $M = (Q, \Sigma, \delta, q0, F)$ and $P_{\Sigma'}(M) = (Q', \Sigma', \delta', p0, F')$ be as in Definition 6. Let $L(M)$ and $L(P_{\Sigma'}(M))$ denote the languages accepted by M and $P_{\Sigma'}(M)$ respectively. Then*

1. *$\forall t \in \Sigma'^*: \delta'^*(p0, t) = p \Rightarrow \forall q \in p, \exists s \in \Sigma^*$ such that $P_{\Sigma'}(s) = t$ and $\delta^*(q0, s) = q$*
2. *$\forall s \in \Sigma^*$ such that $P_{\Sigma'}(s) = t : \delta^*(q0, s) = q \Rightarrow \delta'^*(p0, t) = p$ for some $p \in Q'$, such that $q \in p$.*

Proof (Outline). By induction on the length of s and the fact that $P_{\Sigma'}(s_1 s_2) = P_{\Sigma'}(s_1) P_{\Sigma'}(s_2) \forall s_1, s_2 \in \Sigma^*$. □

Proposition 2. *Given an FSM $M = \{Q, \Sigma, \delta, q0, F)$, let (M) be the language accepted by M. Then $L(P_{\Sigma'}(M)) = P_{\Sigma'}(L(M))$ where $P_{\Sigma'}(M) = (Q', \Sigma', \delta', p0, F')$.*

Proof. Let $s \in L(M)$, then there exists a $q \in F$ such that $\delta^*(q0, s) = q$. Let $P_{\Sigma'}(s) = t$. By Lemma 1, $\delta'^*(p0, t) = p$ for some $p \in Q'$ where $q \in p$. As $q \in p$ and $q \in F$, $p \cap F \neq \emptyset$. Hence $t \in L(P_{\Sigma'}(M))$ and $P_{\Sigma'}(L(M)) \subseteq L(P_{\Sigma'}(M))$. Now let $t \in L(P_{\Sigma'}(M))$, then $\delta'^*(p0, t) = p$ and $p \cap F \neq \emptyset$. Take $q \in p \cap F$, by Lemma 1, there exists $s \in \Sigma^*$, such that $P_{\Sigma'}(s) = t$ and $\delta^*(q0, s) = q$. Therefore $q \in F, s \in L(M)$ and $L(P_{\Sigma'}(M)) \subseteq P_{\Sigma'}(L(M))$. □

3.1. Description of the proposed algorithm

Recall that the submodule construction problem

requires the construction of the specification of an unknown FSM M_2 where the system specification M_0 and the specification of the known submodule M_1 are given as prefix-closed FSMs. We set $M = M_0 \times M_1 = (Q, \Sigma, \delta, q0, F)$ and $\hat{M} = \bar{M}_0 \times M_1 = (\hat{Q}, \hat{\Sigma}, \hat{\delta}, \hat{q}0, \hat{F})$. Note that $Q = \hat{Q}, \Sigma = \hat{\Sigma}, \delta = \hat{\delta}$, and $q0 = \hat{q}0$, however $F \neq \hat{F}$.

A typical state p of M_2 constructed by the proposed algorithm is of the form $p = \{q_1, q_2, \ldots, q_r\}$ where $q_i \in Q$ for all $i \in \{1, 2, \ldots, r\}$.

Definition 7. A state p of M_2 is *valid* if $p \cap F \neq 0$ and $P \cap \hat{F} = \emptyset$.

Algorithm.

input: $M = M_0 \times M_1, \hat{M} = \bar{M}_0 \times M_1, \Sigma_0, \Sigma_1$
output: $M_2 = (Q_2, \Sigma_2, \delta_2, p0_2, F_2)$ if exists and ''No Solution'' otherwise.
begin
$\Sigma_2 := (\Sigma_0 - \Sigma_1) \cup (\Sigma_1 - \Sigma_0), \Delta := (\Sigma_0 \cup \Sigma_1) - \Sigma_2$
$p0_2 := \mathrm{Cls}_\Delta(q0)$
if $p0_2$ is not valid then report ''No Solution'', else
begin
$new := \emptyset$, $old := \{p0_2\}, Q_2 := \{p0_2\}$
while $old \neq \emptyset$ do
begin
for $p \in old$ do
begin
for $\sigma \in \Sigma_2$ do
begin
$p' := \bigcup_{q \in p} \mathrm{Cls}_\Delta(\delta(q, \sigma))$
if $p' \neq \emptyset$ and *valid* then
begin
$\delta_2(p, \sigma) := p'$
if $p' \notin Q_2$ then
begin
$Q_2 := Q_2 \cup \{p'\}$
$new := new \cup \{p'\}$
end
end
else $\delta_2(p, \sigma)$ is undefined
end
end
$old := new$
$new := \emptyset$
end {while}
$F_2 := Q_2$
end
end {Algorithm}.

Remark 1. Note that by construction there are no non-accepting states in M_2 (i.e., $Q_2 = F_2$), indeed all valid states are accepting states and invalid states are discarded by the algorithm. Hence M_2 is prefix-closed.

Proposition 3. Let $M = M_0 \times M_1$, $\hat{M} = \bar{M}_0 \times M_1$, with M_0, M_1 prefix-closed FSMs and M_2 be the output of the proposed algorithm. Then M_2 satisfies the following:

$$L(M_2) = \mathrm{Pre}(P_{\Sigma_2}(L(M)) - P_{\Sigma_2}(L(\hat{M})))$$

Proof. Note that M_2 and $P_{\Sigma_2}(M)$ have the same initial state. Let $s \in \mathrm{Pre}(P_{\Sigma_2}(L(M)) - P_{\Sigma_2}(L(\hat{M})))$ and p be the sate of $P_{\Sigma_2}(M)$ reached after s, then $p \cap F \neq \emptyset$ and $p \cap \hat{F} = \emptyset$. By assumption for all $s' \leq s$, the state p' of $P_{\Sigma_2}(M)$ reached after s' is such that $p' \cap F \neq \emptyset$ and $p' \cap \hat{F} = \emptyset$. By construction of M_2, $\delta_2{}^*(p0_2, s) = p$ and hence $s \in L(M_2)$.

Let $s \in L(M_2)$ then there exists a valid state $p \in Q_2$ such that $\delta_2{}^*(p0_2, s) = p$ also for all $s' \leq s$, $s' \in L(M_2)$ and $\delta_2{}^*(p0_2, s') = p'$ is valid. Hence for all $s' \leq s$, $s' \in P_{\Sigma_2}(L(M)) - P_{\Sigma_2}(L(\hat{M}))$ and therefore $s \in \mathrm{Pre}(P_{\Sigma_2}(L(M)) - P_{\Sigma_2}(L(\hat{M})))$. □

Theorem 2. Given prefix-closed FSMs M_0 and M_1, the FSM M_2 constructed by the proposed algorithm satisfies:

$$P_{\Sigma_0}(L(M_1 \times M_2)) \subseteq L(M_0).$$

Moreover, $L(M_2)$ is maximal with this property.

Proof. Recall the following correspondence: each module is given by a prefix-closed FSM, $\Sigma_i \leftrightarrow V_i$, $L(M_i) \leftrightarrow \mathrm{Pre}(S_i)$, $(i = 0,1,2)$,

$$P_{\Sigma_2}(L(M)) = P_{\Sigma_2}(L(M_0 \times M_1)) \leftrightarrow P_{V_2}(S_0 \times S_1),$$

$$P_{\Sigma_2}(L(\hat{M})) = P_{\Sigma_2}(L(\bar{M}_0 \times M_1)) \leftrightarrow P_{V_2}(\bar{S}_0 \times S_1),$$

$$P_{\Sigma_0}(L(M_1 \times M_2)) \leftrightarrow P_{V_0}(S_1 \times S_2).$$

Note that by Theorem 1, $\mathrm{Pre}(S_2) = \mathrm{Pre}(P_{V_2}(S_0 \times S_1) - P_{V_2}(\bar{S}_0 \times S_1))$ implies $\mathrm{Pre}(P_{V_0}(S_1 \times S_2)) \subseteq \mathrm{Pre}(S_0)$. By the correspondence given above and Proposition 3 we have $\mathrm{Pre}(P_{\Sigma_0}(L(M_1 \times M_2))) \subseteq L(M_0)$. On the other hand, $\mathrm{Pre}(P_{\Sigma_2}(L(M_1 \times M_2))) = P_{\Sigma_2}(L(M_1 \times M_2))$ since M_1 and M_2 are prefix-closed. Therefore $P_{\Sigma_0}(L(M_1 \times M_2)) \subseteq L(M_0)$ and $L(M_2)$ is maximal with this property. □

4. Examples

Each of the three abstract examples given in this section to elaborate the details of the proposed algorithm can be related to a specific domain of application such as

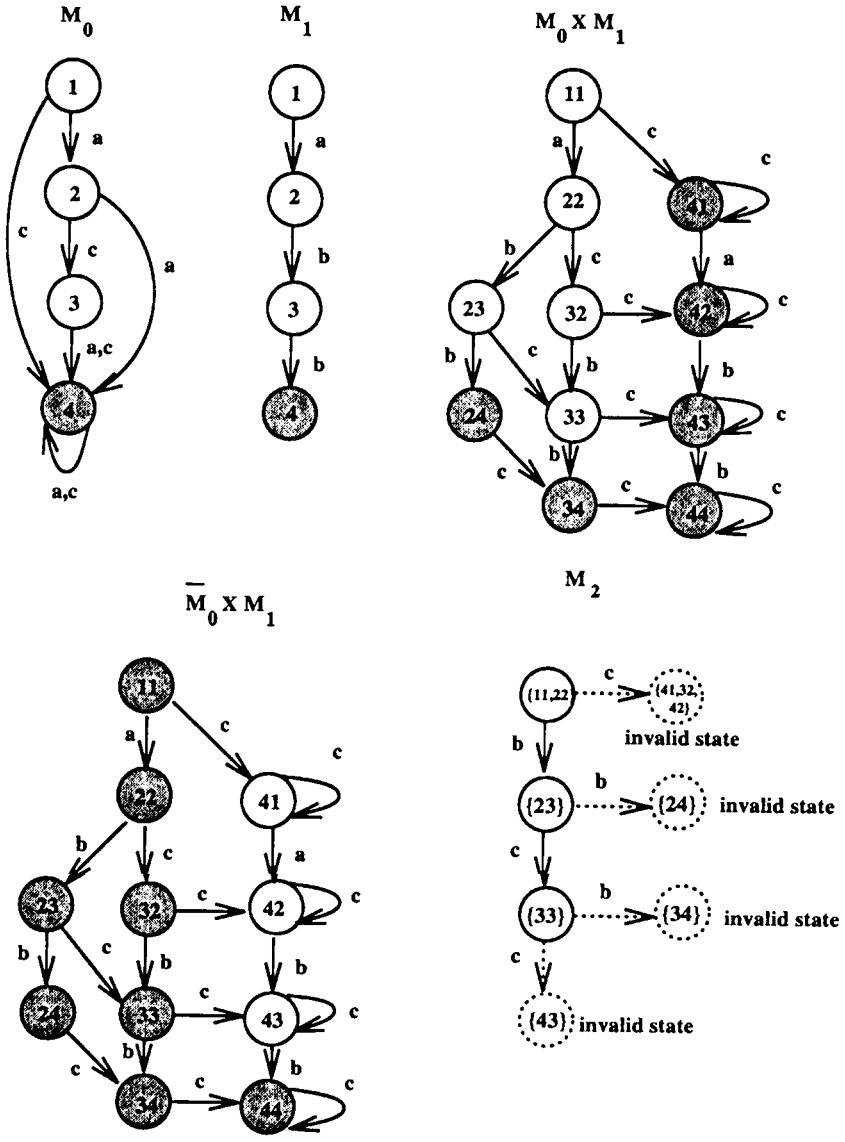

Fig. 2. Example 1.

communication protocols. For example, in the case of Alternating Bit protocol, M_0, M_1, and M_2 stand for the service provided by the protocol, the sender, and the receiver, respectively.

Example 1. In this example we consider the FSMs M_0 and M_1 given in Fig. 2. Both M_0 and M_1 have nonaccepting states, denoted by shaded nodes in Fig. 2. $\Sigma_0 = \{a, c\}$, $\Sigma_1 = \{a, b\}$, and $\Sigma_2 = \{b, c\}$. M_2 produced by the proposed algorithm has 3 valid states. In this case we have $P_{\Sigma_0}(L(M_1 \times M_2)) = L(M_0) = \{\varepsilon, a, ac\}$.

Example 2. Consider the FSMs M_0 and M_1 given in Fig. 3. $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{a, c, d\}$, and $\Sigma_2 = \{b, c, d\}$. M_2 produced by the proposed algorithm has 5 valid states. In this case we have $P_{\Sigma_0}(L(M_1 \times M_2)) = \{\varepsilon, a, ab\}$ and $L(M_0) = (ab)^*(\varepsilon + a)$ and hence, $P_{\Sigma_0}(L(M_1 \times M_2)) \subseteq L(M_0)$. In this example we observe that the FSM M_2 constructed by the proposed algorithm when interacting with M_1 only partially satisfies the specification given by M_0. That is, the set of execution sequences resulting from the interactions between M_1 and M_2 is a proper subset of the set of execution sequences of M_0, $P_{\Sigma_0}(L(M_1 \times M_2)) \subset L(M_0)$.

Example 3 (Special case). In this example we consider the following special instance of the general problem: M_0 and M_1 initially given with all states as accepting states. In this case, a dead state is added to M_0 with necessary

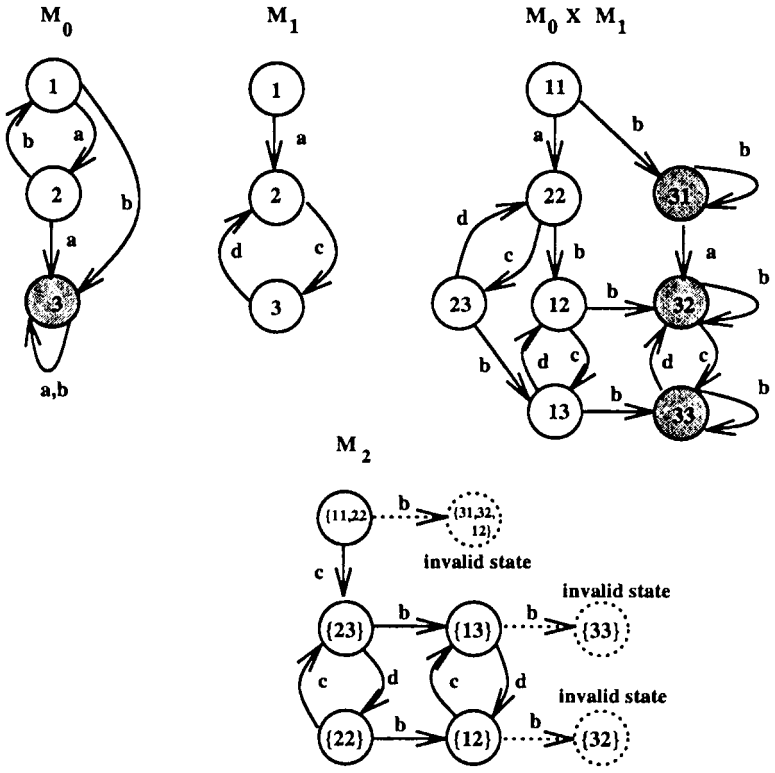

Fig. 3. Example 2.

transitions. This dead state will be the only nonaccepting state of M_0. The algorithm is simplified in this case as it can be easily shown that $\bar{M}_0 \times M_1 = \overline{M_0 \times M_1}$ hence it is sufficient to form only the product $M_0 \times M_1$, since an accepting state of $\bar{M}_0 \times M_1$ is a nonaccepting state of $M_0 \times M_1$. This simplified case covers the treatment presented in [4], as can be seen from the following example adopted from [4] (Fig. 4). In [4] M_0 and M_1 are called BUFFER and 3-CYCLE modules respectively and the ''*a*'' action of M_0 is labelled as ''*put*''. Here, ''*a*'' is used instead of ''*put*'' to force the interaction between M_0 and M_1 on this action.

$\Sigma_0 = \{a, get\}$, $\Sigma_1 = \{a, b, c\}$, and $\Sigma_2 = \{b, c, get\}$. M_2 has 3 valid states. In this case, we have $P_{\Sigma_0}(L(M_1 \times M_2)) = L(M_0) = (a.get)^*(\varepsilon + a)$.

5. Conclusion and related work

We have presented and proven correct an algorithm that finds the most general solution to the submodule construction problem (SCP) where submodules are prefix-closed FSMs with arbitrary number of final states. The FSM M_2, constructed by the algorithm, may not be minimal (i.e., free from redundant transitions and states). It is therefore desirable to devise a mechanism to guarantee that M_2 is minimal. Moreover, as stipulated by the definition of SCP (i.e., find M_2 such that $P_{\Sigma_0}(L(M_1 \times M_2)) \subseteq L(M_0)$), the interactions among M_1 and M_2 may satisfy only a part of M_0 (i.e., the set of execution sequences resulting from the interactions between M_1 and M_2 may be a proper subset of the set of execution sequences of M_0, $P_{\Sigma_0}(L(M_1 \times M_2)) \subset L(M_0)$. Since this case cannot be uniformly avoided in general, it is desirable to establish necessary and sufficient conditions for finding M_2 such that $P_{\Sigma_0}(L(M_1 \times M_2)) = L(M_0)$.

Furthermore, as the formula in [4] and hence our algorithm use the language semantics, they are not sensitive to deadlock and divergence (infinite invisible loop) anomalies. We believe that refining the semantic level to failure equivalence will remedy this shortcoming. This may necessitate the use of refusals set and divergence indices together with the FSM model. Augmented Hoare machine used in [2] may be quite fruitful within this context.

Another possible direction for future work is the extension and formulation of submodule construction problem in the context of communicating finite state machines (CFSMs) [1]. The asynchronous mode of communication between modules in such a framework arises challenging problems. It is the belief of the authors that a proper definition of an asynchronous product seems indispensable as a first step in this treatment. Such an operator will make an algebraic approach possible for the analysis of the communication between submodules (CFSMs).

As stated in Section 1, besides treating the material in [4] in a formal way, our work extends the work of Merlin and Bochmann [4] to handle the general case for the submodule construction problem where each module is considered to be

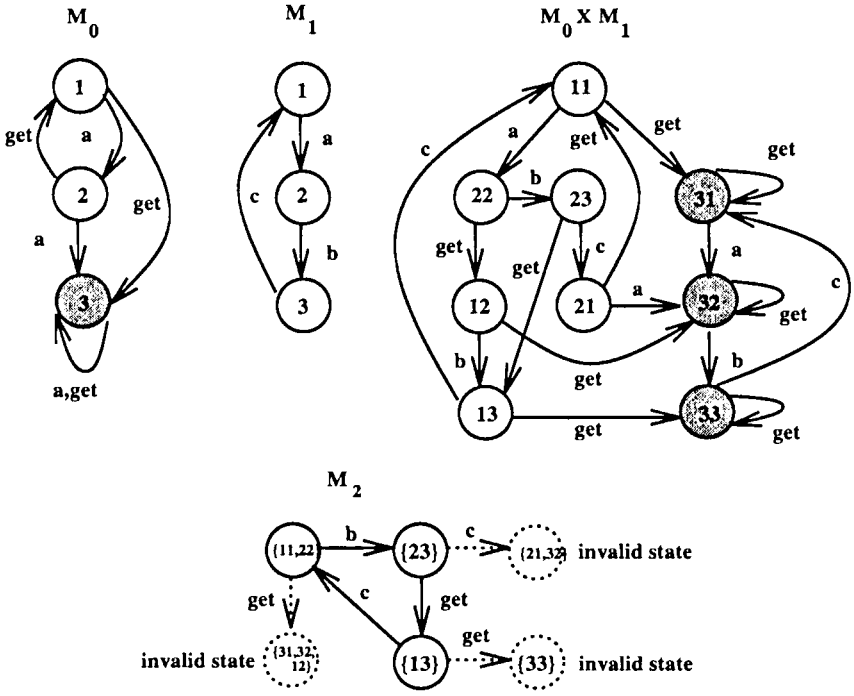

Fig. 4. Example 3.

a prefix-closed FSM (i.e. it accepts a prefix-closed language) with arbitrary number of accepting states, i.e. the submodule construction problem studied in [4] is a special case of the submodule construction problem that is solved by our method. A limitation noted by Merlin and Bochmann was that an automatically generated specification for a submodule may cause deadlocks which stems from the fact that trace equivalence is insensitive to deadlocks.

Several researchers have considered the submodule construction problem using models and notions of equality that are different than the ones studied in [4] and in our work. In [11], the problem is formulated as an equation in CCS [5,6] of the form $(A|X)\backslash L \approx B$ where ''$|$'' and ''$\backslash L$'' are the *composition* and *restriction* operators of *CCS* and model the interaction between the modules and $\approx$ represents the observation equivalence. Necessary and sufficient conditions for the existence of solutions of such equations are given for the case where B is *weakly determinate* and subject to some requirements on the sorts of A and B. An algorithm for constructing a solution is presented. This construction and the requirements for existence of a solution are formulated in terms of the state spaces of A and B.

The work by Qin and Lewis [9] extends the work in [11]. They use a finite state machine model consistent with CCS. An algorithm is presented and proven correct to find the most general solution if solutions exist, where B is deterministic and X is rigid. The key observation remarked by authors is the use of set of pairs of states of A and B to represent a state of X. The algorithm presented in [9] generates states iteratively, beginning from initial state and when an invalid state is encountered a backtracking is necessary to remove incoming links to the state and subsequently the effected states are removed. As a result, extra states are generated and subsequently removed. This overhead can be avoided by detecting an invalid state and by not processing beyond that state. Another shortcoming of this approach is that the algorithm in [9] may fail to produce solutions for the instances of the submodule construction problem which have solutions.

Finally, in [7], the same equation $(A|X)\backslash L \approx B$ is treated and a method is presented for solving such equations where B is deterministic. The method is based on a general tableau framework where a solution is derived via a sequence of transformations. Hence, tableau method formalizes the successive transformational approach to the solution of the equation. These transformations form a basis for the implementation of a semi-automatic program. This program attempts to find a most general solution (a solution which simulates every other solution). The program performs some of the transformations automatically, but a user can effect critical transformations in order to guide the program towards a suitable solution which may be less general but smaller in size. A shortcoming of this approach is that algebraic approaches such as using derivative operators provide a much simpler framework to work with and reason about than that provided by tableau method and their application is less cumbersome than tableau transformations. Moreover, the need for heuristics in guiding the program will then be eliminated since the possibilities of unacceptable solutions are prohibited by the use of rules stipulated by the algebraic approaches.

References

[1] D. Brand, P. Zafiropulo, On communicating finite-state machines, Journal of ACM 30 (2) (1983) 323–342.
[2] E. Haghverdi, K. Inan, Verification by consecutive projections, in: M. Diaz, R. Groz (Eds.), Proceedings of Formal Description Techniques, V (C-10), Elsevier, 1992, pp. 478.
[3] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[4] P. Merlin, G.V. Bochmann, On the construction of submodule specification and communication protocols, ACM Trans. Prog. Lang. Sys. 5 (1) (1983) 1–25.
[5] R. Milner, A calculus of communicating systems, Lecture Notes in Computer Science, 92, Springer, Berlin, 1980.
[6] R. Milner, Communication and Concurrency, Prentice-Hall, 1989.
[7] J. Parrow, Submodule construction as equation solving in CCS, Theoretical Computer Science 68 (1989) 175–202.
[8] R.L. Probert, K. Saleh, Synthesis of communication protocols: survey and assessment, IEEE Trans. on Computers 40 (4) (1991) 468–476.
[9] H. Qin, P. Lewis, Factorization of finite state machines under observation equivalence, Proceedings of International Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France, *Lecture Notes in Computer Science*, 407, Springer, Berlin, 1989.
[10] K. Saleh, Synthesis of communication protocols: an annotated bibliography, ACM SIGCOMM Computer Communications Review 26 (5) (1996) 40–59.
[11] M.W. Shields, Implicit system specification and the interface equation, The Computer Journal 32 (5) (1989) 399–412.