Automated recovery of protocol designs from execution histories*

Jessica Chen
School of Computer Science
University of Windsor
401 Sunset Avenue, Windsor, Canada

Hasan Ural
School of Information Technology and Engineering
University of Ottawa
150 Louis Pasteur, Ottawa, Canada

ABSTRACT

Reverse engineering in distributed systems is an essential activity to recover the designs of large and complex distributed systems that evolve often without proper documentation. In this paper, we present our result in obtaining deadlock-free designs of communication protocols from the observations of the executions of existing systems.

Keyword: Reverse Engineering, Communicating Finite State Machine, Deadlock, Communication Protocols.

1. INTRODUCTION

Software designs are helpful not only for implementing software systems, but also for software maintenance, e.g. to detect and eliminate errors in a system, to extend the capability of a system, or to adapt a system to different operating environments. Moreover, the developers of a new software system whose functionality contains some of the functionality of an existing system can benefit by using the related part of the design of the existing system. However, up-to-date or complete designs of many existing systems may not always be available. This is due to many reasons, e.g. the system may have been poorly documented or the documentation may be either out-of-date or incomplete; the original developers of the system may not be available; or there may be no one who is intimately familiar with the details of the design of the system.

One of the aims of the reverse engineering [4, 8] is to construct the design of an existing system from its implementation. In this paper, we consider the reverse engineering of designs of existing distributed systems, in particular, communication protocols from their implementations. A communication protocol will be modeled by a fixed number of *processes* communicating with each other by sending and receiving *messages* over error-free simplex channels. Each *process* is a protocol entity which is modeled as a *communicating finite state machine* [2, 3] (CFSM for short), and each error-free simplex channel is represented by an unbounded FIFO queue.

The implementation of an existing system from which it is desired to obtain its design may be either source code or executable code. In the case when the source code is used as the input to the reverse engineering process, one can obtain the design by analysing the source code. In this paper, we present our work for the case when the observations of the executions of the executable code are used for obtaining the design of a communication protocol: we start from a given set of observations of the communications among a set of processes, and construct the design of the communication protocol from this set of observations.

Earlier work on the automated construction of designs is focused on the synthesis of designs based on some representation of (partial) requirements [5, 7, 9, 13, among others] or partial designs [15, among others]. These methods relate to the (forward) engineering of the systems rather than their reverse engineering. Among synthesis methods that relate to the reverse engineering are those that consider a given set of observations of an existing implementation [11, 12]. However, these methods start with observations of individual components of the existing system, rearrange events due to some timing considerations, and then utilize a synthesis algorithm. In this paper, we cast the problem of the automated construction of designs in a formal setting, characterize a given set of observations of the implementation of a system, and demonstrate the relationship between the design constructed by our proposed method to the presumed design implied by the given set of observations.

The designs obtained from the requirements through a manual process usually contain errors such as deadlocks. Similarly, the designs obtained through an automated process may also contain these types of errors [15]. Therefore, either a manual intervention is needed to eliminate the errors [15], or the automated identification of all errors need to be carried out via some formal analysis [6, 10, 15] on the constructed design. In this paper, we show that determinization of the constructed design not only reduces the number of states in the component CFSMs but also can reduce some deadlocks. Yet it is not always possible to eliminate all the deadlocks only by determinization. Hence, we propose a technique for the automated deadlock elimination. This technique is based on using some additional observations that are implied by the given set of observations and thus preserves the semantics of the system viewed by an external observer.

*This work is partially supported by The Natural Sciences and Engineering Research Council of Canada under grant numbers STR 0149338, OGP 0000976, and OGP 0209774.

The rest of the paper is organized as follows: Section 2 introduces the terminology and notation used throughout the paper. In Section 3, we show the construction of the design of a system from a given set of observations. Section 4 addresses the issue of minimization of the constructed designs with possible determinization applied. In Section 5, we present a technique to obtain deadlock-free designs by utilizing the additional (implied) observations. We give our concluding remarks in the last section.

2. PRELIMINARIES

Definition 2.1 *A communicating finite state machine is a quadruple $(S, M, s^0, \rightarrow)$ where S is a finite set of states and $s^0 \in S$ is the initial state; M is a finite set of messages; $\rightarrow \subseteq S \times \{+m, -m \mid m \in M\} \times S$ is a set of transitions.*

A transition $(s, \mu, s') \in \rightarrow$ of a process, also denoted as $s \xrightarrow{\mu} s'$, has the intuition that the process changes its state from s to s' by event μ. We use $-m$ to denote the event of sending message m, and $+m$ the event of receiving message m. Moveover, we will use E_M to denote the set of events of sending/receiving messages in M, i.e. $E_M = \{+, -\} \times M$, and we use $\mu,\ \mu_1,\ \mu_2, \ldots$ to range over it.

In a protocol, we use binary relation (i, j) to denote the existence of a simplex channel from process i to process j, and we use $M_{i,j}$ to denote all messages that can be put onto it. For convenience, we assume that the messages in different channels are all distinct.

Definition 2.2 *A communication protocol P is a triple (L, M, T) where*

1. *$L \subseteq \{(i, j) \mid 1 \leq i, j \leq n, i \neq j\}$ for $n \geq 2$;*
2. *$M = \{M_{i,j} \mid (i, j) \in L\}$ with $M_{i,j} \cap M_{k,l} = \emptyset$ if $(i, j) \neq (k, l)$;*
3. *$T = \{(S_i, M_i, s_i^0, \rightarrow_i) \mid 1 \leq i \leq n,$
   $M_i = \bigcup_{j=1,\ldots,n, j\neq i}(M_{i,j} \cup M_{j,i})\}$ is a set of CFSMs.*

The (global) state of a protocol is composed of a (local) state of each process and a content of each channel. The evolution of a protocol is described in terms of the transitions from one (global) state to another. Such transitions are built up on the basis of the transitions of each process, taking into account their effects on the contents of related channels. We use a (possibly empty) string of messages to denote the content of a channel. We use $|\omega|$ to denote the length of a string ω and $\omega[i]$ ($|\omega| \geq i$) to denote the ith element of ω.

Initially, each process is in its initial state, and all channels are empty, denoted by empty string ε. Thus, the initial (global) state is the state where all processes are in their initial states, and all channels are empty.

Definition 2.3 *The network N of protocol $P = (L, \{M_{i,j} \mid (i, j) \in L\}, \{(S_i, M_i, s_i^0, \rightarrow_i)\}_{i=1}^n)$ is a quadruple $(S, M, s^0, \rightarrow)$, where*

1. *$S = \Pi_{i=1,\ldots,n} S_i \times \Pi_{(i,j)\in L} M_{i,j}^*$;*
2. *$M = \bigcup_{i=1,\ldots,n} M_i$;*
3. *$s^0 = \Pi_{i=1,\ldots,n} s_i^0 \times \Pi_{(i,j)\in L}\ \varepsilon$; (i.e., initial global state)*
4. *$\rightarrow \subseteq S \times E_M \times S$ is the set of transitions.*

Definition 2.4 *Let $N = (S, M, s^0, \rightarrow)$ be a network of protocol P. A state $s \in S$ is called reachable if $s = s_0$ or for some $r \geq 1$, $\exists\, s^1, \ldots, s^r \in S$, $\exists \mu_1, \ldots, \mu_r \in E_M$ s.t. $s^r = s$ and $s^{i-1} \xrightarrow{\mu_i} s^i$ for $i = 1, \ldots, r$.*

We will use R to denote the reachable global state space of a protocol P. Clearly, R of $P \subseteq S$ of N of P. Moreover, let s be a reachable state, we will use the following notations:

- $s^1 \xrightarrow{\mu_1\mu_2\ldots\mu_r} s^{r+1}$ if $\exists s^2, \ldots, s^r$ s.t. $s^i \xrightarrow{\mu_i} s^{i+1}$ for $i = 1, \ldots, r$;
- $s \xrightarrow{\sigma}$ where $\sigma \in E_M^*$ if $\exists s'$ $s.t.$ $s \xrightarrow{\sigma} s'$;
- $s \not\xrightarrow{\sigma}$ where $\sigma \in E_M^*$ if $\nexists s'$ $s.t.$ $s \xrightarrow{\sigma} s'$.

A *deadlock* is a reachable global state where all channels are empty and no process can send a message. In this paper, we consider deadlock states. We will use *error state* for a shorthand of deadlock state. We will call a protocol or its design *error-free* if none of the states in its reachable global state space is an error state.

3. CONSTRUCTING DESIGNS

By monitoring the behavior of the implementation of a protocol during its execution, we can observe (or deduce) a (possibly infinite) sequence of events (i.e., transmission or reception of messages), ordered according to their time of occurrences. The global observations reflect the functionality of a communication protocol in terms of sequences of events that occur during the execution of the protocol. We assume that the functionality of the implementation is periodic, i.e. global observations start from, and end at the initial global state. We also assume that the executions are monitored by someone who is familiar with the functinality of the implementation, so that the periodicity can be recognized.

Definition 3.1 *A global observation is a sequence of events of a protocol that starts from and ends at the initial global state without crossing over the initial state of any of the component processes.*

Given a global observation o, we can derive $proj(o,i)$, the projection of o on process i:

$$proj(o,i) = \begin{cases} \varepsilon & o = \varepsilon \\ -m_{i,j}proj(o',i) & o = -m_{i,j}o' \\ & \text{for } 1 \leq j \leq n \\ +m_{j,i}proj(o',i) & o = +m_{j,i}o' \\ & \text{for } 1 \leq j \leq n \\ proj(o',i) & o = -m_{l,j}o' \text{ or } +m_{j,l}o' \\ & \text{for } 1 \leq l,j \leq n,\ l \neq i \end{cases}$$

Here we use $+m_{i,j}$ to denote the observation of the event that process j receives message m from process i, and $-m_{i,j}$ the observation of the event that process i sends message m to process j. $proj(o,i)$ reflects the sequence of events within the global observation o that are related to process i. It can be shown that for any global observation o and for any i, $proj(o,i)$ starts from and ends at the initial state of process i without going over the initial state of process i.

Note that $proj(o,i)$ is analogous to a unilogue [14] of process i in the presumed design. Moreover, when the number of the processes in the protocol is two, $(proj(o,1),\ proj(o,2))$ is in fact, a duologue [14] in the presumed design, and o is simply an occurrence (i.e., successful execution) of this duologue. Furthermore, when the number of processes in the protocol is greater than two, $(proj(o,1),\ldots,\ proj(o,n))$ is a multilogue [1] in the presumed design.

Consequently, given a nonempty set $O = \{o_1,\ \ldots,\ o_r\}$ of global observations, we have set $proj(O,\ i)$ of unilogues over O for process i:

$$proj(O,\ i) = \{proj(o_j,\ i) \mid 1 \leq j \leq r,\ proj(o_j,\ i) \neq \varepsilon\}$$

We shall construct the protocol design starting from a given set of global observations. Without loss of generality, we will asume that the given set of observations does not contain empty strings. Suppose that an implemetation of a protocol P, consisting of n processes, and a set O of global observations of this implementation are given. Let $PD(O)$ denote the error-free presumed design that is implied by the given set O of global observations and $seq(P)$ denote the set of all occurrences (i.e., successful executions) of multilogues in P. Then the problem that will be studied in this paper is to derive a constructed design $CD(O)$ such that $CD(O)$ is error-free and $seq(CD(O)) \subseteq seq(PD(O))$. For a solution of this problem, we use $proj(O,\ i)$ of unilogues over O for process i, for $i = 1,\ldots,n$, to construct the CFSM of process i.

Definition 3.2 *Given a set $proj(O,i) = \{b_1,\ \ldots,\ b_k\}$ $(k \geq 1)$ of unilogues over O for process i, where $b_l \neq \varepsilon$ for $l = 1,\ldots,k$. The observation generated CFSM over $proj(O,i)$ is $(S, M_i, s^0, \rightarrow)$ where*

1. $S = \{s^{l,j} \mid 1 \leq l \leq k,\ 1 \leq j \leq |b_l| - 1\} \cup \{s^0\}$*;*
2. $s_0 \in S$ *is the initial state;*
3. $M_i = \bigcup_{j=1,\ldots,n, j\neq i}(M_{i,j} \cup M_{j,i})$
4. $\rightarrow \subseteq S \times E_{M_i} \times S$ *is the least relation satisfying:*
   (a) $s^0 \xrightarrow{b_l[1]} s^{l,1}$ *for* $1 \leq l \leq k$*;*
   (b) $s^{l,j-1} \xrightarrow{b_l[j]} s^{l,j}$ *for* $1 \leq l \leq k,\ 2 \leq j \leq |b_l| - 1$*;*
   (c) $s^{l,j-1} \xrightarrow{b_l[j]} s^0$ *for* $1 \leq l \leq k,\ j = |b_l| \geq 2$*;*
   (d) $s^0 \xrightarrow{b_l[1]} s^0$ *for* $1 \leq l \leq k,\ |b_l| = 1$.

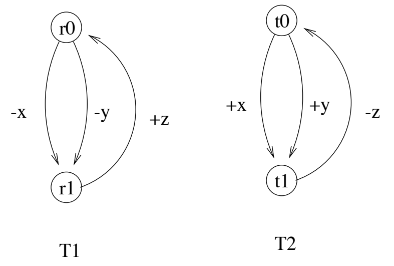

Figure 1. A presumed design

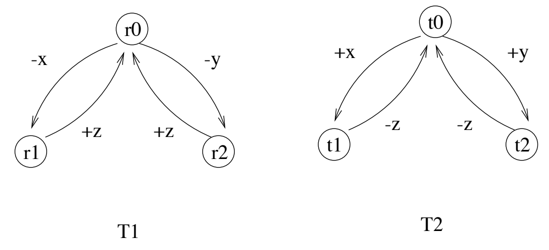

Figure 2. A constructed design CD_0 from global observations

Let L be a set of channels among n processes, M a set of legal messages exchanged over the channels in L, and O a given set of global observations. In the following, we use $CD_0(O)$ to denote the constructed design $(L, M, \{T_i\}_{i=1}^n)$ where for $i = 1,\ldots,n$, T_i is the observation generated CFSM over $proj(O,i)$.

Example 3.1 Consider a protocol with two processes P_1 and P_2 and two channels between them. Let the given set of global observations obtained from the implementation of the protocol be:

$$O = \{-x_{1,2} + x_{1,2} - z_{2,1} + z_{2,1},\\ -y_{1,2} + y_{1,2} - z_{2,1} + z_{2,1}\}$$

Figure 1 and Figure 2 show the presumed design $PD(O)$ and the constructed design $CD_0(O)$, respectively. Here, T_1 and T_2 in $CD_0(O)$ are the CFSMs over $proj(O,1)$ and $proj(O,2)$ respectively.

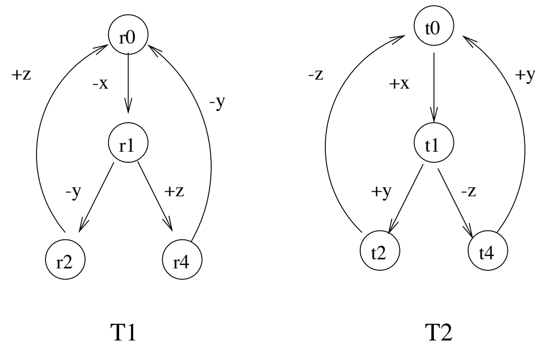

Figure 3. A presumed design

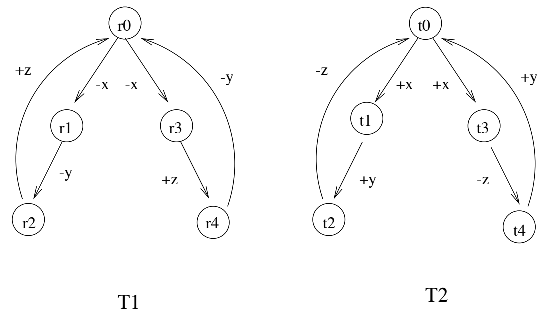

Figure 4. A constructed design

Like the designs constructed by a manual process from the requirements, the designs constructed through an automated process from a given set of observations may contain design errors. Within this context, it is important to note that $CD_0(O)$ may contain errors even if $PD(O)$ is error-free. There are two sources that cause the errors in $CD_0(O)$: the nondeterminism of $CD_0(O)$ and the incompleteness of the given set O of global observations. The example below shows a typical instance where the errors in $CD_0(O)$ come from the nondeterminism of the CFSMs in $CD_0(O)$.

Example 3.2 Consider a protocol with two processes P_1 and P_2 and two channels between them. Figure 3 shows the presumed design of this protocol. From the existing implementation of this protocol, suppose that we obtained

$$O = \{-x_{1,2} + x_{1,2} - z_{2,1} + z_{2,1} - y_{1,2} + y_{1,2} \\ -x_{1,2} + x_{1,2} - y_{1,2} + y_{1,2} - z_{2,1} + z_{2,1}\}$$

The constructed design shown in Figure 4 contains a deadlock: If P_1 sends x to P_2 entering state r^3, and P_2 receives this message and enters state t^1 then the global state reached $(r^3, t^1, \epsilon, \epsilon)$ is a deadlock state. By eliminating the non-determinism in the CFSMs of the constructed design, we obtain the protocol in Figure 3 which is deadlock free.

In the following, we discuss the situation when the error-freeness in the presumed design implies the error-freeness in the constructed design. As we have mentioned, there are two sources which cause the presence of errors in the constructed design even if the presumed design is error-free: nondeterminism and incompleteness of observations. The latter can cause the errors in the constructed design only when there are collisions in the presumed design. Here, collision means that two processes may send messages concurrently. Some applications, typically the Client/Server models, are without collisions. Below, we show that, if the presumed design is free from collision, i.e. at each global state, there is only one process that is able to send messages, then the constructed design is free from errors as long as it is deterministic.

Let σ be a sequence of events. The negation of σ, denoted by $\bar{\sigma}$, is defined as below:

$$\bar{\sigma} = \begin{cases} \varepsilon & \sigma = \varepsilon \\ -m\overline{\sigma'} & \sigma = +m\sigma' \\ +m\overline{\sigma'} & \sigma = -m\sigma' \end{cases}$$

It can be shown that if $PD(O)$ is error-free and free from collisions, and the constructed design $CD(O)$ is deterministic, then $CD(O)$ is free from errors. When $PD(O)$ is not free from collisions, it is possible that the presumed design is error-free while the deterministic constructed design contains errors. That is, let $CD(O)$ be a deterministic constructed design. If $PD(O)$ is error-free while $CD(O)$ contains an error state, then (1) there exists a unilogue which belongs to a process of $PD(O)$ but not in the same process of $CD(O)$; (2) there exists an execution sequence which can be obtained from $PD(O)$ but which does not belong to O.

4. MINIMIZING DESIGNS

In this section, we introduce three algorithms to improve this originally constructed design in terms of reducing the number of states of its CFSMs and in terms of reducing (or if possible, eliminating) the deadlock states. We will use *constructed design* for either $CD_0(O)$ or any design derived from it by applying algorithms introduced in this paper.

Algorithm A is the determinization of the CFSMs. Note that in the CFSMs in $CD_0(O)$, except for the initial state, each state has exactly one incoming edge. We call this kind of CFSM *tree-like* CFSM. This algorithm applies only to these tree-like structures. It is easy to see that it takes $\mathcal{O}(v \times e^2)$ time to apply Algorithm A to a CFSM T in $CD_0(O)$. Here v is the number of states in T, and e is the maximum number of outgoing edges from the states in T. In the following, we use $G_A(P)$ to denote the protocol derived from P by applying Algorithm A. By applying Algorithm A to a constructed design with tree-like CFSMs, we do not introduce new unilogues into any of the CFSMs in $CD_0(O)$, thus we do not change the set of occurrences of the multilogues. Thus, we have (1) $seq(G_A(P)) =$

$seq(P)$ for any protocol P which contains only tree-like CFSMs; (2) $seq(G_A(CD_0(O))) \subseteq seq(PD(O))$.

As we have mentioned, Algorithm A can help removing error states. An example is that the reachable global state space of the protocol in Figure 4 contains a deadlock state $(r^3, t^1, \epsilon, \epsilon)$. By applying Algorithm A to the CFSMs, we obtain the protocol in Figure 3 which is deadlock free.

While Algorithm A sometimes help reducing error states, it does not introduce new errors to tree-like constructed designs.

The Algorithm B reduces the number of states in the constructed design but, can only apply to the so-called mono-historic CFSMs as defined below.

Let $pos(\omega)$ denote the sequence of all the events of receiving messages in ω in the same order, and $neg(\omega)$ the sequence of all the events of sending messages in ω in the same order. I.e.

$$pos(\omega) = \begin{cases} \varepsilon & \omega = \varepsilon \\ +m\, pos(\omega') & \omega = +m\omega' \\ pos(\omega') & \omega = -m\omega' \end{cases}$$

$$neg(\omega) = \begin{cases} \varepsilon & \omega = \varepsilon \\ -m\, neg(\omega') & \omega = -m\omega' \\ neg(\omega') & \omega = +m\omega' \end{cases}$$

Definition 4.1 *A CFSM T is called mono-historic if for any non-initial state s in T, and for any sequence of events between s^0 and s (without passing s^0 twice), $pos(\sigma)$ and $neg(\sigma)$ are unique.*

In a mono-historic CFSM, we use $neg(s)$ and $pos(s)$ to denote the sequences of sending and receiving events respectively from s^0 to s (without passing s^0 twice). Let v be the number of states in the CFSM, m be the maximum number of states i-step reachable from s_0 for all $1 \leq i \leq l$. Then we need to compare $\mathcal{O}(v \times m)$ pair of states since we compare only those sates that have same number of steps from s_0. Each comparison takes constant time, so the total cost of running this algorithm is $\mathcal{O}(v \times m)$.

The CFSMs in $CD_0(O)$ are obviously mono-historic, so we can apply Algorithm B to them. It is easy to see that, Algorithm B is more general than Algorithm A when we consider mono-historic CFSMs. We will use $G_B(P)$ to denote the protocol derived from P by applying Algorithm B.

Example 4.1 Again, suppose we have two processes P_1 and P_2 and two channels between them. Let

$o_1 = -x_{1,2} - y_{2,1} + x_{1,2} - z_{1,2} + y_{2,1} + z_{1,2} - u_{2,1} + u_{2,1} - v_{1,2} + v_{1,2}$

$o_2 = -x_{1,2} + x_{1,2} - y_{2,1} + y_{2,1} - z_{1,2} + z_{1,2} - v_{1,2} + v_{1,2} - u_{2,1} + u_{2,1}$

$O = \{o_1,\, o_2\}$

Figure 5 and 6 shows the $PD(O)$ and $G_B(CD_0(O))$ respectively.

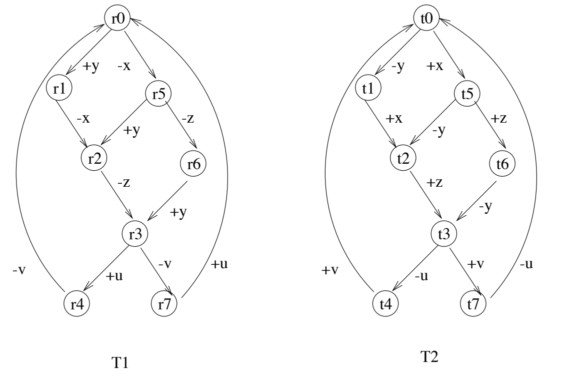

Figure 5. A presumed design

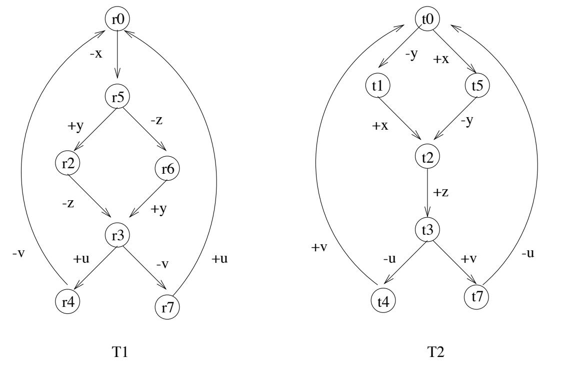

Figure 6. A constructed design by applying Algorithm B

Unlike Algorithm A, Algorithm B may introduce new execution sequences to the constructed design. That is, in general, $seq(G_B(CD_0(O))) \not\subseteq seq(PD(O))$. This is because, by merging two states according to Algorithm B, we may add new unilogues.

It is observed that we may reduce the number of states in $CD_0(O)$ in addition to the reduction obtained by Algorithm A, i.e. $G_A(CD_0(O))$. To apply Algorithm C to a CFSM T in $G_A(CD_0(O))$, we need to consider each pair (s^1, s^2) of states in T, and compare each outgoing edge from s^1 with each outgoing edge from s^2. So totally it takes $\mathcal{O}(v^2 \times e^2)$ time to apply Algorithm C to T. Here v is the number of states in T, and e is the maximum number of outgoing edges from the states in T.

In the following, we use $G_C(P)$ to denote the protocol derived from P by applying Algorithm C.

Similar as Algorithm A, Algorithm C can be applied to the constructed design without introducing new execution sequences. That is, (1) $seq(G_C(P)) = seq(P)$; (2) $seq(G_C(G_A(CD_0(O)))) \subseteq seq(PD(O))$. Thus, Algorithm C does not contribute to the removal of any error, nor does it introduce new errors to the constructed design.

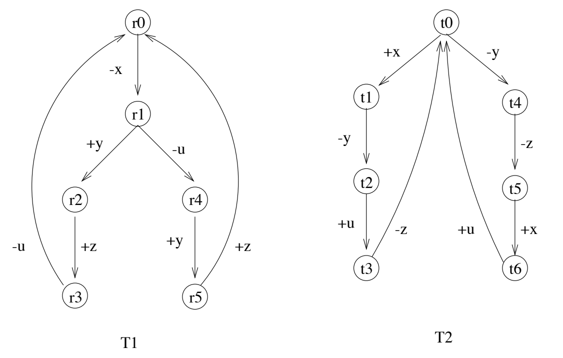

Figure 7. A constructed design with a deadlock state

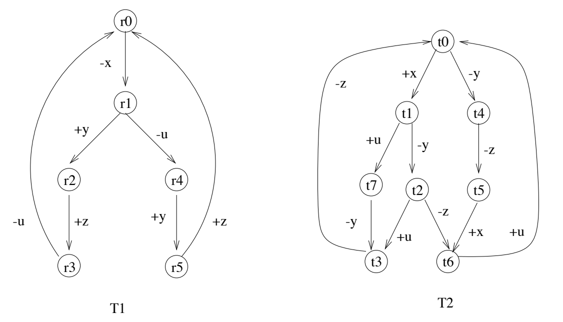

Figure 8. A protocol obtained by applying Algorithm D

5. ELIMINATING ERRORS

If the presumed design is not free from collisions, the constructed design derived by applying the above algorithms may still contain errors. Figure 7 shows a constructed design with the Algorithms A and C no more applicable. It can be easily seen that the reachable global state space of this protocol contains a deadlock state i.e., $(r^2, t^2, \epsilon, \epsilon)$. According to what we have discussed so far, since Algorithm A has been applied, such errors can only come from the incompleteness of the observations. So we need to execute the implementation again to collect more global observations.

Alternatively, we can also provide some algorithms to modify the constructed design. Rather than the previous three algorithms which mainly intend to reduce the number of states in the constructed design, here we would like to have some algorithms to eliminate the errors.

Algorithm D uses the negation of the derived unilogues to eliminate the deadlocks in the constructed design. It can only apply to two-process protocols with mono-historic CFSMs. Note that $CD_0(O)$, $G_A(CD_0(O))$ and $G_B(CD_0(O))$ contain only mono-historic CFSMs.

We will use $G_D(P)$ to denote the protocol obtained from P by applying Algorithm D.

Example 5.1 Figure 8 shows the protocol derived by applying Algorithm D on the protocol in Figure 7. Note that while the protocol in Figure 7 contains a deadlock, the protocol in Figure 8 is free from errors.

6. CONCLUSIONS AND FINAL REMARKS

We have shown that the determinization and minimization of the designs constructed from a given set of execution histories may not be sufficient to eliminate all the dedlocks. Thus, we have presented Algorithm D which can eliminate all the deadlocks in the constructed design. That is, for any constructed design P with only mono-historic CFSMs, $G_D(P)$ is deadlock-free.

Note that the application of Algorithm D adds some unilogues to the constructed design. These unilogues may or may not be present in the presumed design. So generally speaking, Algorithm D may add occurrences of new multilogues in $seq(G_D(P))$. These executions may have never been included in the given set of observations obtained from the implementation of P. However, a good design very often includes the negations of all the unilogues of one CFSM into the other CFSM, because a unilogue and its negation in two distinct processes usually reflect one of the correct functionalities of the protocol.

References

[1] N. Arakawa and T. Soneoka. A test case generation method for concurrent programs. in: Proc. of IWPTS'91, Leidschendam, The Netherlands, pp.95-106, 1991.

[2] G. v. Bochmann. Finite state descriptions of communication protocols. *Computer Networks*, 2: 361–372, 1978.

[3] D. Brand, and P. Zafiropulo. On communicating finite state machines. *J. ACM*, 30(2): 323–342, 1983.

[4] E. Chikofsky, and J. Cross. Reverse Engineering and design recovery. *IEEE Software*, 7: 13-17, Jan. 1990.

[5] T.Y. Choi, and R.E. Miller. A decomposition method for the analysis and design of finite state protocols. *ACM SIGOMM'83*, pp. 167–176, 1983.

[6] G.J. Holzmann. Design and Validation of Computer Protocols. Prentice Hall, 1991.

[7] K. Koskimies, and E. Makinen. Automatic synthesis of state machines from trace diagrams. *Software Practice and Experience*, 24(7): 643-658, 1994.

[8] D. Lee and K. Sabnani. Reverse engineering of communication protocols. *Proc. of IEEE ICNP'93*, pp. 208-216, 1993.

[9] M. Rajagopal, and R.E. Miller. Synthesizing a protocol converter from executable protocol traces. *IEEE Trans. on Computers*, 40(4): 487-499, 1991.

[10] J. Rubin, and C.H. West. An improved protocol validation technique. *Computer Networks and ISDN Systems*, 6: 65-73, 1982.

[11] K. Saleh, and A. Baujarwah. Communications software reverse engineering. *J. of Information and Software Technology*, 38: 379-390, 1996.

[12] K. Saleh, R.L. Probert, and I. Manonmani. Recovery of communication protocol design from protocol execution traces. *IEEE ICECCS'96*, pp. 265-272, Montreal, PQ, Oct. 1996.

[13] S. Some, R. Dssouli, and J. Vaucher. From scenarios to timed-automata: Building specifications from user requirements. *APSEC'95*, Dec. 1995.

[14] P. Zafiropulo. Protocol validation by duologue-matrix analysis. *IEEE Trans. on Commun.*, 26(8): 1187–1194, 1978.

[15] P. Zafiropulo, C.H. West, H. Rudin, D.D. Cowan, and D. Brand. Towards Analying and Synthesizing Protocols. *IEEE Trans. on Commun.*, 28(4): 651–660, 1980.