Protocol Conversion between Complex Protocols*

J.C. Shu and Ming T. Liu

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210-1277

ABSTRACT

With the proliferation of different computer networks, protocol conversion is needed to achieve interoperability between computer networks that implement different protocols, and to resolve the incompatibility between protocols so that users on different networks can communicate with each other. It is not practical to convert all messages one-to-one from one protocol to another because of various protocol mismatches. For complex protocols, it is even possible that the messages produced in some order, by one protocol, need to be converted into the messages in a different order, for the other protocol. This is called indirect conversion, for which the produced messages are stored in a non-FIFO buffer, and then converted. Okumura [1] has proposed a conversion seed method to specify the semantic relationship between the messages of two given protocols. Based on the conversion seed, a single-process converter is constructed from the given protocols. However, the resulting converter itself does not reveal concurrence, and may lack some required functionality such as data retransmission. Lam [2] has proposed a formal model to find a semantically common image of two protocols, and then the rest of the required functionality is added to complete the construction of a converter. However, the performance of this approach is very sensitive to how much common functionality these two protocols have. Furthermore, neither Okumura nor Lam has addressed the problem of indirect conversion.

In our previous work [3], we have proposed a formal model to synthesize a converter by inserting synchronization points into two given protocols. Each synchronization point consists of a pair of asynchronous communication operations PUT and GET. The PUT operation inserts a message into a non-FIFO buffer, and then the GET operation retrieves and converts the message in the buffer. The PUT and GET operations may create some problems for the buffer: (1) channel overflow, i.e., the number of messages stored into the buffer exceeds its capacity; (2) improper termination, i.e., the protocol terminates with some message left over in the buffer. Therefore, the resultant converter must be validated so that it is free from protocol errors, such as channel overflow and improper termination. Several techniques are also proposed to optimize the computing time and space spent in validating a converter.

1. Introduction

Users on different computer networks cannot easily communicate with each other due to the proliferation of different network architectures and communication protocols. Protocol conversion is needed to resolve the incompatibility between different protocols so that they can interoperate between different networks. First of all, semantic equivalence between the messages of two protocols has to be identified; and then a converter is constructed to convert the messages of one protocol into semantically equivalent messages for the other protocol. It is not practical to convert all messages one-to-one from one protocol to the other because of various protocol mismatches. For complex protocols, it is even possible that the messages produced in some order, by one protocol, need to be converted into the messages in a different order, for the other protocol. This is called indirect conversion, for which the produced messages are stored in a non-FIFO buffer, and then converted.

Okumura [1] has proposed a conversion seed method to specify the semantic relationship between the messages of two given protocols. Based on the conversion seed, a single-process converter is constructed from the given protocols. However, the resulting converter itself does not reveal concurrence, and may lack some required functionality such as data retransmission. Lam [2] has proposed a formal model to find a semantically common image of two protocols, and then the rest of the required functionality is added to complete the construction of a converter. However, the performance of this approach is very sensitive to how much common functionality these two protocols have. Furthermore, neither Okumura nor Lam has addressed the problem of indirect conversion.

In our previous work [3], we have proposed a formal model to synthesize a converter by inserting synchronization points into two given protocols. Each synchronization point consists of a pair of asynchronous communication operations PUT and GET. The PUT operation inserts a message into a non-FIFO buffer, and then the GET operation retrieves and converts the message in the buffer. However, the PUT and GET operations may create some problems for the buffer: (1) channel overflow, i.e., the number of messages stored into the buffer exceeds its capacity; (2) improper termination, i.e., the protocol terminates with some message left over in the buffer. Therefore, the resultant converter must be validated so that it is free from protocol errors, such as channel overflow and improper termination. Several techniques are also proposed to optimize the computing time and space spent in validating a converter.

1. Introduction

Users on different computer networks cannot easily communicate with each other due to the proliferation of different network architectures and communication protocols. Protocol conversion is needed to resolve the incompatibility between different protocols so that they can interoperate between different networks. First of all, semantic equivalence between the messages of two protocols has to be identified; and then a converter is constructed to convert the messages of one protocol into semantically equivalent messages for the other protocol. It is not practical to convert all messages one-to-one from one

*Research reported herein was supported by U.S. Army CECOM, Ft. Monmouth, NJ, under Contract No. DAA560-88-K-0003. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

1 A conversion seed is a finite state machine to specify the rule for the occurrence of messages in two protocols, and gives the guidelines and properties of converter construction.
One of the most effective ways to validate communication protocols is reachability analysis that exhaustively explores all possible interactions of communication protocol entities. The interactions happening in protocol conversion can be classified into two categories: intra-protocol and inter-protocol. The intra-protocol interaction occurs between the entities in the same protocol, and the inter-protocol interaction in different protocols. Inter-protocol interactions are introduced by inserting synchronization points into the given protocols, so that messages are buffered, converted, and passed from one protocol to the other. Under the assumption that two given protocols are valid, the exploration of intra-protocol interactions can be bypassed in the reachability analysis so as to optimize computing time and space.

The rest of this paper is organized as follows. Section 2 introduces basic definitions, and a converter is defined on the basis of process communication. In Section 3, section centers on validating a converter, and presents some optimization techniques for reachability analysis. Finally, some concluding remarks are given in Section 5.

2. The Model

In our model, each protocol consists of a pair of processes, and each process is represented by a communicating finite state machine (CFSM). A CFSM is a 5-tuple \((S, s_0, M, \delta, F)\), where \(S\) is a set of states, \(s_0\) (in \(S\)) is the initial state, \(M\) is a set of messages, \(\delta\) (contained by \(S\)) is the set of final states, and \(\delta\) is the transition function that is naturally extended to: \(\delta(s, e) = s\) and \(\delta(s, Hm) = \delta(s, H, m)\), where \(m\) is a message and \(H\) is a message sequence.

Including the set \(F\) in the definition of CFSMs has an advantage: a complex CFSM can be partitioned into smaller pieces, with each piece being an individual CFSM. Consequently, the problem of verifying a property preserved by a CFSM often can be decomposed into the sub-problems of verifying that property on smaller CFSMs separately.

In what follows, we shall give an overview of basic definitions, to be used in this paper, by using some examples. The reader is referred to [3] for additional definitions and details.

In Figure 1, we have two protocols \(P = [P_s, P_r]\) and \(Q = [Q_s, Q_r]\), where \(P_s, P_r, Q_s,\) and \(Q_r\) are CFSMs. The minus sign on an edge indicates the transmission of a message, and the plus sign means its reception. \(P\) is a simple SEND-ACK protocol, in which \(P_s\) transmits \(D\) to \(P_r\), and \(P_r\) responds with \(ACK\) as acknowledgement. On the other hand, \(Q\) is the Alternating Bit Protoc...
Q's shown in Figure 2, a new composite protocol \([P, P', Q, Q']\) is formed to allow \(P\) and \(Q\) to communicate. In the same way, two composite protocols are allowed to form another composite protocol; thus, the general form of a protocol \(P\) is \([P, P_1, \ldots, P_n]\), in which each pair of \(P_i\) and \(P_j\), for \(3 \leq i \leq n\), is an embedded converter, \(P_i + P_j\). We assume that the communication taking place within a simple protocol is asynchronous via non-FIFO buffers, and that taking place across protocols is asynchronous via non-FIFO buffers. So, each CFSM in a protocol is associated with an implicit reception queue, and each CFSM in a converter is associated with an implicit reception buffer. Note that for a protocol \(P\) with \(n\) CFSMs, \(P = \{P_1, P_2, \ldots, P_n\}\), there are \(n\) reception queues and \(n - 2\) reception buffers.

The behavior of a protocol is characterized in terms of global states. A global state \(G\) of protocol \([P_1, P_2, \ldots, P_n]\) is \(\{(s_1, e_1, e_2), \ldots, (s_n, e_n)\}\), where \(s_i\) is the current state of \(P_i\), \(b_i\) is a message sequence representing the contents of \(P_i\)'s reception buffer (non-FIFO), and \(q_i\) is a message sequence representing the contents of \(P_i\)'s reception queue (FIFO). Global state \(G\) is the next state of \(G'\), if \(i, k, q_i\), and \(q_k\) such that all elements of \(G\) and \(G'\) are the same except

- \(s_i = b_i(\text{first message}) + m\) and \(q_i = m_q\), or
- \(s_i = b_i(\text{last message}) - m\) and \(q_i = m_q\), or
- \(s_i = b_i(\text{first message}) + m\) and \(b_i = m\), or
- \(s_i = b_i(\text{last message}) - m\) and \(b_i = m\), and \(m\) is in \(b_i\), where

\ (+m\) represents \(\text{GET}(m)\) as mentioned earlier, \(-m\) represents \(\text{PUT}(m)\), and \((+m)\) means deleting \(m\) from \(b_i\).

In the previous example of protocol \([P, P', Q', Q']\), \(G_0 = \{(0,0,0,0),[e, e, e, e]\}\) is the initial global state, and an execution of the protocol could be:

- \(G_0 \vdash G_1 = \{(1,0,0,0),[e, e, D, e, e]\}\)
- \(G_1 \vdash G_2 = \{(1,2,0,0),[e, e, e, e, e]\}\)
- \(G_2 \vdash G_3 = \{(1,1,0,0),[e, e, D, e, e]\}\)
- \(G_3 \vdash G_4 = \{(1,1,4,0),[e, e, e, e, e]\}\)
- \(G_4 \vdash G_5 = \{(1,1,1,0),[e, e, e, e, e]\}\)
- \(G_5 \vdash G_6 = \{(1,1,1,1),[e, e, e, e, e]\}\)
- \(G_6 \vdash G_7 = \{(1,1,2,2),[e, e, e, e, e]\}\)

In this execution, the message sequence \(H\) generated by \(P\) is \(\pm D + A0/A1 - ACK + D - D0 + A1 - A0\), and that by \(Q\) is \(\pm D0 + A0/A1 + D - D0 + A1 - A0\). Global states \(G_0\) through \(G_9\) are reachable from \(G_0\). In addition, transition \(S(p_0, -D)\) is executable in \(G_0\), while \(S(q_0, 0, D0)\) is executable in \(G_5\).

3. Constructing a Converter

A converter is intended to carry out the mapping specified in a mapping set. There are as many potential converters as there are possible mapping functions between the messages of two protocols. However, only those converters which make two protocols interoperable are acceptable; i.e., their composite protocols should be free from deadlock, unexpected reception, channel overflow, and improper termination.

Definition 3.1. Let \(G = (\{s_1, s_2, \ldots, s_n\}, \{b_1, b_2, \ldots, b_m\}, \{q_1, q_2, \ldots, q_p\})\) be a reachable global state of protocol \([P, P_1, \ldots, P_n]\), then the protocol is said to be:

1. Free from deadlock iff \(\forall G, \exists s\) and \(x\), such that \(\delta(s, x)\) is executable.
2. Free from unexpected reception iff \(\forall G\) and \(i\), \(\delta(s_i, +m)\) is executable in \(P_i\), when the first message in \(q_i\) is \(m\).
3. Free from channel overflow iff \(\forall G\) and \(i\), \(|q_i| \leq b_i\), and \(|b_i| \leq C_i\) where \(C_i\) and \(b_i\) are the sizes of \(q_i\) and \(b_i\), respectively.
4. Free from improper termination iff \(\forall G\) and \(i\), \(q_i\) and \(b_i\) are empty, provided that \(\forall k, s_k \in F_k\).

Moreover, a protocol is said to be valid iff it is free from deadlock, unexpected reception, channel overflow, and improper termination.

We allow two composite protocols to form another new composite protocol; thus, the new protocol can be a long sequence of CF-MSMs. In order to simplify the notation, we give protocols a general representation: \([P, P', Q, Q']\) or \([P, P]\), where \(P\) or \(P\) is a boundary CFSM in the protocol to be used for synthesizing a converter, and \(P\) is a sequence of the other CFSMs.

Definition 3.2. Let \(sr(H)\) represent the message sequence obtained by dropping all buffer messages from \(H\), and \(pg(H)\) be the sequence obtained from \(H\) by dropping those messages which are not buffer messages. A converter \(C = Pr + Qs\) with mapping set \(T\) for protocols \(P = [P, P']\) and \(Q = [Q, Q']\) is valid iff the following conditions hold for the composite protocol \(R = [P, P', Q, Q']\):

1. \(H\) is generated by \(Pr'\) (in an execution of \(R\)) only if \(sr(H)\) is generated by \(Pr\) (in an execution of \(P\)). Also, \(H\) is generated by \(Qs\) only if \(sr(H)\) is generated by \(Qs\).
2. In an execution of \(R\), if \(Hp\) and \(Hq\) are generated by \(Pr'\) and \(Qs\) respectively, then \(pg(Hp)\) is a rearrangement of \(pg(Hq)\), where the rearrangement means that the messages' positions in the sequence can be arbitrarily switched.
3. \(R\) is valid.

The first condition means that \(C\) satisfies the syntactic requirements of the original protocols; that is, \(C\) behaves as \(Pr\) for \(P\), and as \(Qs\) for \(Q\). The second condition assures that \(C\) meets the semantic requirements of the conversion; that is, \(C\) can carry out the mapping specified in \(T\). If the buffers used by the converter were FIFO, the \(pg(Hp)\) would be almost the same as \(pg(Hq)\) rather than the rearrangement as mentioned above. Furthermore, for the converters with FIFO buffers, the semantic requirements could be verified individually between each pair of consecutively generated \(p\)-messages or \(q\)-messages, because the first \(\text{GET}\) executed must be on the first message that has been \(\text{PUT}\) into the buffer. However, this decomposition for the problem of semantic verification is pre-

![Figure 2: A converter for the protocols in Figure 1.](image-url)
It is worth noting that if two GETs are to be inserted before the same -message, then they are inserted in parallel, such as GET(A0) and GET(A1) in Figure 2. If a PUT and a GET are to be inserted before the same -message, then they are inserted serially with the PUT preceding the GET, such as PUT(A0) and GET(D) in Figure 2. On the other hand, if two Puts are to be inserted before the same -message, they are inserted serially in an arbitrary order.

Theorem 1. Inserting PUTs and GETs before -messages for the construction of a converter satisfies the first condition in Definition 3.2.

Proof
Assume that $Pr' + Qs'$ is a converter, and message sequence $H$ is generated by $Pr'$ in an execution of $\{P, Pr', Qs', Q\}$. When any buffer message $m$ is generated by $Pr$ in the execution, the -message following $m$ in $H$ can also be generated by $Pr$ in the corresponding execution (without PUTs and GETs) of $\{P, Pr\}$. Since $sr(H)$ is exactly the sequence obtained by dropping those buffer messages from $H$, $sr(H)$ can be generated by $Pr'$.

On the other hand, if $H$ is generated by $Qs'$ instead of $Pr'$, then by the same reason as above, $sr(H)$ can be generated by $Qs'$.

Once a mapping set is specified for two protocols, the construction of the converter is mechanized by inserting $PUT \cdot GET$ pairs into the protocols. Up to this point, what we can say about the resultant converter is, as stated in Theorem 1, that the converter meets the syntactic requirements. The next section will discuss how to validate the converter so as to satisfy the other conditions in Definition 3.2.

4. Validating a Converter

A valid converter must make two protocols interoperable; i.e., their composite protocol should be valid. In order to verify that the protocol is valid, constructing a reachability graph is one of the most effective ways, because the retrieval of buffer messages in an execution is totally unpredictable. The reachability graph is constructed by successively exploring all reachable global states from the initial global state. However, the construction of reachability graphs often encounters a problem: state space explosion (the size of the graph grows beyond the memory space). To alleviate this state explosion problem, one optimization is to reduce the size of those global states stored in the graph. This optimization is achievable under the assumption that the original protocols are valid, and will be proved as correct in Theorem 2. In the following definition, a converter state is extracted from a global state, and will play the global state's role in the reachability graph.

Definition 4.1. Assume that $C = P \cdot P'$ is an embedded converter in protocol $P = \{P_1, \ldots, P_n\}$, and $G = \{s_1, s_2, \ldots, s_n\}, \{r_1, r_2, \ldots, r_{n-1}\}, \{q_1, q_2, \ldots, q_m\}$ is a global state of $P$. The converter state $g$ of $C$ for global state $G$ is a sub-sequence of $G$, $(s, s_1, b, h, q)$.

Definition 4.2. The initial converter state, denoted as $g_0$, is when $P_1$ and $P_2$ are in their initial states, and both $b$ and $q$ are empty. A final converter state is when both $P_1$ and $P_2$ are in their final states.

Converter states allow us to alleviate the state explosion problem, but can they be used to speed up the construction of the reachability graph? Yes, the speedup can be achieved by using converter states jointly with follow sets. The follow set reveals, given a state in a CFSM, which buffer operations (PUTs and GETs) will possibly be executed next.
Definition 4.3. The follow set of state $s$ in a CFSM, denoted as $\text{follow}(s)$, is the set $\{b \mid (s, H, b) \in \text{head}(s) \land H \text{ contains no buffer messages}\}$.

If we set up follow sets before the construction of the reachability graph, a significant speedup can be achieved during the construction, because the trace of consecutively executable transitions in the CFSMs can be omitted. For example, in Figure 2, $\text{follow}(1)$ in $Pr'$ is $\{+A0, +A1\}$, and $\text{follow}(4)$ in $Qx$ is $\{-A0\}$, where $+A0$ represents $\text{PUT}(A0)$ and $-A0$ represents $\text{PUT}(A0)$, as mentioned in Section 2. If the converter state is $(1, 4, c, e)$, then instead of tracing a sequence of executable transitions in $Pr'$ or $Qx$, we know that there exists some $H$ such that $(s, (4, H), -A0) = \text{tail}(-A0) = 2$. Therefore, the next converter state of our concern, $(1, 2, A0/4)$, can be derived solely from $\text{follow}(4)$.

Definition 4.4. Assume that $g = (s_a, s_b, b_p, b_h)$ is a converter state of $Pr' + Qx$. Then $g'$ is said to be the next state of $g$, denoted as $g \rightarrow g'$, iff for either $i = p$ and $j = q$, or $i = q$ and $j = p$.

1. $x \in \text{follow}(s_b)$ and $\text{head}(x) = s_i$, then $s_i = \text{head}(x), s_j = s_j$.
2. $s_b = b_b$, and $b_h = b_h$.
3. if $m_i \in \text{follow}(s_b)$ and $m_i$ is in $b_h$, then $s_i = \text{tail}(m_i), s_j = s_j$.
4. $b_h = b_h$ ($m_i$ deleting $m_i$), and $b_b = b_b$.
5. if $m_i \in \text{follow}(s_b)$, then $s_i = \text{tail}(m_i), s_j = s_j, s_b = b_b m_i$, and $b_h = b_h$.

Definition 4.5. If $g \rightarrow g'$ and if $P_i$ changes from state $s_i$ in $g$ to state $s'_i$ in $g'$ by making transition $b_i(s_i, H)$, then transition $b_i(s_i, H)$ is said to be executable in $g$. Also, a converter state $g_a$ is reachable iff $g_0 \rightarrow g_1 \rightarrow \ldots \rightarrow g_a$.

During the construction of the reachability graph, a path in the graph may be terminated with a final converter state or a detected error. A detected error represents a deadlock, channel overflow, or improper termination.

Definition 4.6. Converter state $g = (s_p, s_q, b_p, b_q)$ of $Pr' + Qx$ encounters

1. a deadlock iff there does not exist any executable transition in $g$.
2. a channel overflow if the number of messages in $b_p$ or $b_q$ exceeds the buffer's capacity.
3. an improper termination iff $s_p$ and $s_q$ are the final states of $Pr'$ and $Qx$, respectively, but $b_p$ or $b_q$ is not empty.

Exploring converter states instead of global states in the reachability graph greatly alleviates the state explosion problem, because a lot of information on queues and states need not be traced. However, we can further optimize the size of the reachability graph by reducing the number of converter states saved in the graph. In fact, only those converter states providing useful information, such as a suspected deadlock or channel overflow, need be saved. Thus, a converter state $g$ of $Pr' + Qx$ is saved, only if one of the following rules can be applied.

1. $s$ is in $g$, and $m_i/j \in \text{follow}(s)$, but $m_i$ is not available in the buffer (a potential deadlock).
2. $s$ is in $g$, and $-m_i/j \in \text{follow}(s)$, but the reception buffer is full (a channel overflow).
3. $g$ is a final converter state (a potential improper termination).
4. A PUT operation $-m_i/j$ has been executed, and $\text{follow}(\text{tail}(-m_i/j))$ contains only GET operations (a potential maximum number of messages in the buffer).

The above four rules guide the construction of the reachability graph. Rules 1 through 3 are self-explanatory. The protocol designer often wants to know the maximum number of messages accumulated in the buffer during an execution. Rule 4 is proper for this purpose, because if $\text{follow}(\text{tail}(-m_i/j))$ contains a PUT operation, then this PUT can be immediately executed, and the number of messages in the buffer is incremented. If the converter ends up to be valid, the maximum number of messages in the buffers can be used to determine their sizes for implementation.

Figure 5: A converter $Pr' + Qx$.
Theorem 2. A converter $C = Pr' + Qs'$ for protocols $P = (Pr, Pr')$ and $Q = (Qs, Q)$ is valid if both $P$ and $Q$ are valid, and for any reachable converter state $g$, $g$ does not encounter any deadlock, channel overflow, or improper termination.

Proof

The first condition in Definition 3.2 is satisfied as proved in Theorem 1. We will show that the other conditions are satisfied by counter proof. Let $R = [R_1, R_2, \ldots, R_n]$ represent the composite protocol $[P, Pr', Qs', Q]$. We first prove the "if" part of this theorem, and then the "only if" part.

If converter $C$ does not satisfy the second condition in Definition 3.2, then there must be some buffer message $z$, such that $z$ is in $pg(H)$ but not in $pg(H')$, where $i = p$ and $j = q$, or $i = q$ and $j = p$. Without loss of generality, we choose $i = p$ and $j = q$. Let $z$ be a $g$-message, generated by $Pr'$, then $z$ stays in $H$ through the execution. Thus, an improper termination is encountered. On the other hand, let $z$ be a $g$-message, $z$ would have been generated by $Qs'$, and then saved in $H'$, so that $Pr'$ could perform GET($z$) for generating $z$. This contradicts the assumption that $z$ is not in $pg(H)$.

If $R$ is not free from deadlock, then there must be a reachable global state $G$, in which no transition is executable. Theorem 1 and Definition 4.4 together imply that the converter state $g$ for $G$ is reachable iff $G$ is reachable. Thus $g$ is reachable and with no executable transition; i.e. $g$ encounters a deadlock.

If $R$ is not free from improper termination, then there are some reachable global state $G$ and $k$, such that all $R_k$ are in their final states, but $q_k$ or $b_k$ is not empty. Assume that $P$ and $Q$ are valid, then by Theorem 1, $q_k$ is empty for all reachable $G$ and $k$. Thus, the only cause of improper termination is that $b_k$ is not empty for some $k$. This implies that for some reachable $g$, $b_k$ is not empty when both $Pr'$ and $Qs'$ are in their final states; i.e. $g$ encounters an improper termination.

Similarly as in the proof for improper termination, if $R$ is not free from channel overflow, we can conclude that some reachable converter state $g$ encounters a channel overflow. On the other hand, if $R$ is not free from unexpected reception, then by Theorem 1 and the way that $Pr'$ and $Qs'$ are inserted, $P$ or $Q$ must not be free from unexpected reception; i.e. $P$ or $Q$ is not valid.

For the "only if" part of this theorem, we assume that $P$ and $Q$ are valid. Since the converter state $g$ for global state $G$ is reachable iff $G$ is reachable, it is obvious that any deadlock, channel overflow, or improper termination encountered by a reachable $g$ reflects that $C$ does not meet the semantic requirements, or $R$ is not free from deadlock, channel overflow, or improper termination.

Theorem 2 shows that if there is no error detected in the reachability graph, then the synthesized converter is valid. However, it is often the case that some errors will show up. Figure 7 shows an example of a channel overflow, which is frequently found in validating a converter. $Pr'$ keeps on inserting $m_1$ into the buffer, and $Qs'$ in turn removes $m_1$ from the buffer. If there is a speed mismatch between $Pr'$ and $Qs'$ such that $Pr'$ is faster, then the number of messages in the buffer will grow out of bound; i.e. a channel overflow occurs. This kind of errors is normally fixed by adding a GET to the faster CFSM, so that it can be synchronized by the other CFSM. Thus, in Figure 7, if we include $<u > +$ in the mapping set, the overflow error goes away.

Figure 8 is another example, in which a deadlock can occur. In an execution, $Pr'$ can generate $-u + m_1 - m_1$ and reach state 5, while $Qs'$ generates $+y$ and gets stuck in state 3. The converter state becomes ($5', 3', 5, m_1$), and encounters a deadlock. This situation occurs because $Qs'$ cannot be enforced to make transition $8(0, +z)$, when $Pr'$ makes the transition $8(0, -u)$. However, this unpleasant result can be remedied by including $-z, +y >$ in the mapping set.
Most of time, an improper termination occurs in the form as shown in Figure 9. There is a chance that \( Pr \) generates \(+m-\) and reaches the final state 2, while \( Qr \) generates \(-U\) and terminates. The converter state becomes \((2, 2', E, m/)\), where 2 and 2' are final states, and one buffer is not empty. Therefore, an improper termination is encountered. Adding \(<+U, +>\) in the mapping set will fix this error.

Figure 9: A converter with an improper termination.

5. Conclusion

The proliferation of network architectures and protocols has made protocol conversion a very important design problem in computer communications. Various protocol mismatches complicate the design of a general algorithm for protocol conversion, and preclude one-to-one mapping between all messages of two protocols. This paper presents formal techniques to analyze indirect conversion for mapping only the significant messages between two protocols, and proposes an asynchronous PUT-GET mechanism to construct a converter with non-FIFO buffers. In order to verify that the converter is valid, constructing a reachability graph is one of the most effective ways, because the retrieval of buffer messages in an execution is totally unpredictable.

Under the assumption that two given protocols are valid, converter states can be used to construct a reachability graph for validating a synthesized converter, thereby achieving two optimizations: (1) reducing the size of each state in the reachability graph, and (2) reducing the number of states stored in the graph. Follow sets can be easily set up before the construction of the reachability graph, and then significantly speed up the construction. In addition, those protocol errors most often revealed in the reachability graph are described, and their fixes are made. The resultant converter is composed of individual CFSMs that represent concurrent processes, and their inherent concurrency allows the converter to work more efficiently.

References