Synthesizing Converters between Finite State Protocols

Janaki Akella

Dept. of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh PA 15213

Kenneth McMillan

School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213

Abstract

Inter-process communication within computer systems is becoming increasingly important due to the recent trend of integrating various types of subsystems and system architectures. These subsystems may not have been originally designed to work with one another, therefore their integration constitutes a heterogenous computer system, where even small mismatches in protocols will prevent proper operation of the system. Standardization of protocols is both impractical and sub-optimal with respect to any particular subsystem because of the numerous and often conflicting requirements. Thus inter-process communication can be achieved only by protocol conversion. No general theory for synthesizing protocol conversions exists, though there have been several successful specific protocol conversions. In this paper we present a general approach for synthesizing protocol conversions, by adapting existing labelled transition system theory.

1 Introduction

Inter-process communication within computer systems is becoming increasingly important due to the recent trend of integrating various types of subsystems and system architectures. In the past, heterogeneous computer networks were used, but each computer was itself homogeneous in that all its subsystems followed the same protocol. But the proliferation of specialized applications such as input/output and networking has resulted in the need to integrate various types of subsystems within the computer system [1]. These subsystems may not have been originally designed to work with one another, therefore their integration constitutes a heterogeneous computer system, where even small mismatches in protocols will prevent proper operation of the system [5]. Standardization of protocols is both impractical and sub-optimal because of the numerous and often conflicting requirements. Thus inter-process communication can be achieved only by protocol conversion. No general theory for synthesizing protocol conversions exists, though there have been several successful specific protocol conversions. In such cases, the inter-process communication devices (IPCDs) such as bus adapters, gateways, and bridges convert one communication protocol to another so that transactions begun on one bus can end on another with minimum changes required in the hardware and software modules of the two buses. In this paper we propose a general approach for synthesizing IPCDs by adapting Labelled Transition Systems (LTS) [7]. Several studies [2] show that the required rate of protocol conversion will in general be much less than the maximum rate of information processing in the interface subsystems. Thus the additional time taken for protocol conversion may not be prohibitive.

Given the protocols of the interfaces that need to communicate with one another, several issues arise in automating the design of the protocol converter. These issues can be classified into: information Transfer issues, synchronization issues, timing issues, concurrency issues, and observability issues. In this work, we propose an approach to automate the generation of the state machine of the protocol converter, i.e., given the interface protocols which are represented by finite state machines, we propose an approach to generate the finite state machine representing the protocol converter. We assume that the data path of the protocol converter is already given.

1.1 Related Work

Although little work was done in the past on automatically synthesizing inter-process communications devices, recently work has been done on two related problems: synthesis of digital systems with interfaces [3], and synthesis of transducers which interface custom chips to their system buses [10]. Nestor's work deals with synthesizing the digital system together with the interface from a ISPS (Instruction Set Processor Specifications) description, when the communication protocols are matched. Timing labels attached to relevant statements were used to calculate inter-event timing, to arrive at a schedule of events. Here, the sequence of events between the digital system and its interface is indirectly obtained from the timing information, with the consequent problem that the timing between every pair of events is required to determine their sequence. Borriello's work deals with synthesizing a transducer between synchronous as well as asynchronous protocols from the corresponding timing diagrams. Merge labels and ordering labels attached at appropriate points of the event sequences on each side of the transducer determine the order of execution of the events in the transducer. Thus the sequence of events in the inter-process communication is explicitly marked with the effect that the all possible orderings

CH3040-3/91/0000/0410$01.00 © 1991 IEEE

of events between the processes are not modeled.

Other existing formalisms that have been used to model interface processes either have a notion of sequencing and primarily model asynchronous communication (basic Petri Nets, and path expressions [9]) or have a notion of sequencing but not timing and model only synchronous communication (CCS [6]), or model sequencing and timing but mainly model synchronous communication (finite state machines [8]). In this work, we use labelled transition systems in which the labels are divided into two types *input labels* and *output labels*, to model sequencing, typical timing constraints such as minimum set-up and hold times, deskew and decode times, asynchronous communication (i.e., all possible event orderings) and synchronous communication. Double synchronization of the signals can also be modeled by introducing additional states in the LTS. In Section 2 we describe the generation of the communicating process in detail.

The proposed LTS modeling and LTS product are modifications to the parallel composition of synchronous Moore machines, [4]. This work has been used to model large synchronous systems to verify their correctness by making assertions of safety and liveness in temporal logic. But this cannot be directly applied to model communication between different protocols as it requires that both machines jointly make transitions at every clock step. When the protocols are such that the two machines do not have corresponding transitions then this composition results in error states.

2 Proposed Approach

The two different interface protocols that need to communicate with one another are specified by their corresponding finite state machines. Each protocol has an output side (the actions performed when the interface is to a master device) and an input side (the actions when the interface is to a slave device). The signals from the output side are observable by the protocol converter, and the signals to the input side are controllable by the protocol converter. All interface transitions caused by internal signals are neither observable nor controllable by the protocol converter. Correct operation between the output side of one protocol and the input side of the second protocol requires that the outputs and the inputs occur at the expected times. A failure occurs if the first protocol produces an output when the second protocol is not expecting it. The problem is to synthesize the protocol converter using the observable and controllable signals such that the two interfaces together with the protocol converter operate correctly. For example, consider an interface to a four-phase protocol, and an interface to a two-phase protocol that need to communicate with one another. Figure 1 shows the timing diagrams of a four phase protocol and a two phase protocol. It can be seen that if these two interfaces were to be directly connected then the two phase protocol interface begins its second transaction even before the four phase protocol interface has completed the first transaction resulting in an error.

In this work the finite state machine part of the

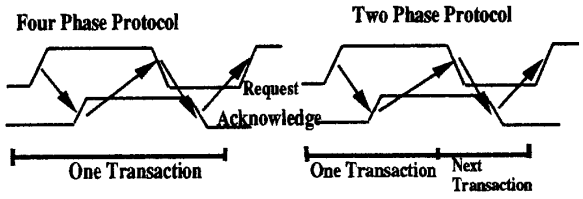

Figure 1: Timing diagrams of Two-phase and Four-phase protocols

protocol converter is obtained similar to the product of the general labelled transition systems (LTSs) [7]. Clearly this would result in a large number of states even for simple LTSs. The product LTS is reduced by four steps: firstly, a third LTS C which represents the desired sequence of inter-operation between the interfaces is specified, and the product of the interface LTSs is jointly obtained with the C machine. This is defined below and illustrated by the powerset composition in Figures 3, 4, and 5 in the example in Section 3. The resulting LTS includes all the undesired sequences of inter-operations as well. Therefore the second step is to eliminate these undesired sequences. Further reduction is obtained by a third step, in which the transitions that are caused by signals not controllable by the protocol converter are eliminated. The final step consists in hiding signals unobservable by the protocol converter. Since the protocol converter cannot observe signals internal to each interface process, the transitions caused by the internal signals are removed and the corresponding present and next states are merged. This hiding of the unobservable signals results in a determinized LTS where the transitions are between sets of states. The number of states in each state set can be reasonably expected to be between 2 and 3, because the interface controller seldom performs extensive data manipulation. This determinizing process (hiding of unobservable signals) is shown in Figure 6 for the example in Section 3. The choice of transitions in the determinized LTS is guided by progress in the C machine.

A general LTS is a structure given by the four-tuple*(S, i. L, Tran)* where:

S is a set of states with initial state i

L is a set of labels,

that do not contain the symbol '*'

$Tran \subseteq S \times L \times S$

is the transition relation

To model the interfaces, the LTS labels are divided into two types- *input labels* and *output labels*. The transitions in the communicating process are generated by obtaining the product of the interface LTSs, with the LTS of the C machine. The resuting product

includes all possible interleavings of the transition labels. Thus either machine singly making a transition as well both machines jointly making a transition are modeled.

Their product of two general LTSs is given next. Let T_0 and T_1 be two LTSs, their product $T_0 \times T_1$ is defined below.

$$\begin{aligned}
T_0 &= (S_0, i_0, L_0, Tran_0) \\
T_1 &= (S_1, i_1, L_1, Tran_1) \\
T_0 \times T_1 &= (S, i, L, Tran) \\
&where \\
S &= S_0 \times S_1 \\
i &= i_0 \times i_1, \\
&The projections: \\
\rho_0 &= S_0 \times S_1 \rightarrow S_0 \\
\rho_1 &= S_0 \times S_1 \rightarrow S_1 \\
L &= L_0 \times L_1 = \\
(\alpha, *) \mid \alpha \in L_0 \cup & \\
(*, \beta) \mid \beta \in L_1 \cup & \\
(\alpha\beta) \mid \alpha \in L_0 and \beta \in L_1 & \\
&with projections \\
(s, \alpha, s') &\in Tran_* \Leftrightarrow \\
(\rho_0(s), \pi_0(\alpha), \rho_0(s')) &\in Tran_{0*}, \text{and} \\
(\rho_1 s), \pi_1(\alpha), \rho_1(s')) &\in Tran_{1*}.
\end{aligned}$$

$Tran_{0*}$, and $Tran_{1*}$ are transition relations including the *idle transition*, where a idle transition (analogous to a self-loop in FSMs) is given by $(s, *, s) \mid s \in S$

Intuitively, this includes transitions with the labels of the form α, β representing synchronizations between two processes set in parallel, transitions with the labels of the form $*, \beta$ or $\alpha, *$ representing transitions in only one process, unsynchronized with the other process. Clearly this composition results in an explosion in the number of states, as all possible interleavings of the labels are to be included. The C machine is given by a LTS T_2. Any operation sequence not specified by T_2 results implicitly in an error state. The product $T_0 \times T_1 \times T_2$ is obtained and the illegal transitions and transition not controllable by the communicating process are eliminated.

Section 3 describes protocol converter generation.

3 Example

Figure 2 shows the LTS representation corresponding to the protocols shown in Figure 1. The 'exleft' interface is to a slave device following a two phase protocol. and the 'exright' interface is to master device following a four phase protocol.

The LTS given by the exleft machine is:

$$\begin{aligned}
T_0 &= (S_0, i_0, L_0, Tran_0) \\
&where \\
S_0 &= 0, 1, 2 \\
i_0 &= 0
\end{aligned}$$

r := Req
a := Ack
l := infoin , infout

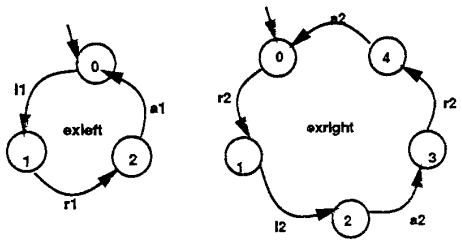

Figure 2: LTS representation of Two-phase and Four-phase protocols

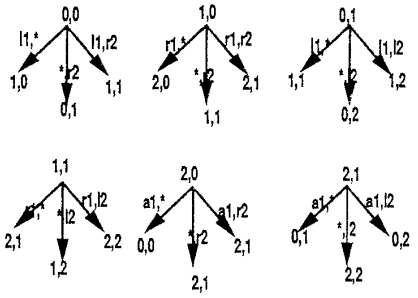

Figure 3: Product of the two LTSs

$$\begin{aligned}
L_0 &= a_1, l_1, r_1 \\
Tran_0 &\subset S_0 \times L_0 \times S_0
\end{aligned}$$

and the LTS given by the exright machine is:

$$\begin{aligned}
T_1 &= (S_1, i_1, L_1, Tran_1) \\
&where \\
S_1 &= 0.1.2.3.4 \\
i_1 &= 0 \\
L_1 &= r_2, l_2, a_2 \\
Tran_1 &\subset S_1 \times L_1 \times S_1
\end{aligned}$$

Part of the result of step 1 of the product of exleft and exright machines is given in Figure 3. Each cluster represents a joint present state, possible interleavings of transitions between the two machines, and the possible joint next states. The remaining steps are similar.

The C machine representing the correct interoperation between the two machines is given below

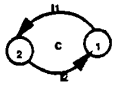

Figure 4: Correct inter-operation between the LTSs

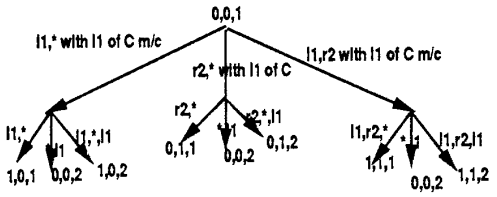

Figure 5: Part of the Product of exleft, exright and C machines

in Figure 4.

The next states in the product machine from the joint state 0,0,1 is shown below in Figure 5. The complete product can be obtained in a similar manner.

All possible inter-process communication is obtained by the product of exleft, exright, and the C machine. The reduction of the resulting LTS is obtained by eliminating illegal transitions and transitions not controllable by the communicating process. Signals a_1 and r_2 are outputs of the communicating process, and therefore are controllable. Signals r_1, and a_2 are inputs to the communicating process, and signals l_1, l_2, are internal to machines exleft and exright. Transition conditions such as l_1,*,l_1 and l_2,*,l_2 represent conditions where an internal transition is reflected by a corresponding transition in the C machine and therefore represents a valid condition. Conditions such as l_1, * and l_2, where an internal transition was made but not reflected by the C, machine are invalid. Therefore of the possible states from the joint present state of 0,0,1 shown in Figure 5, the possible next states are 0,1,1 and 1,0,2. In a similar manner the complete product can be reduced.

This reduced LTS includes transitions not visible to the communicating process.

We have presented a new approach for synthesizing the communication process given the interface process specification and illustrated the approach by generating the communication process between a four phase master and a two phase slave. Work is currently under way to synthesize coverters between popular backplane buses.

We would like to express our appreciation to Edmund Clarke and David Long for the useful discussions that helped to clarify the issues involved in interface modeling and in protocol converter generation.

```
((R1 A2)
 (A1 R2)
 ((((0 0 1) (1 0 2)) (R1 (2 0 2)))
  (((2 0 2)) (R2 (2 1 2) (2 2 1)))
  (((2 1 2) (2 2 1)) (A2 (2 3 1)))
  (((2 3 1)) (A1 (0 3 1) (1 3 2)) (R2 (2 4 1)))
  (((0 3 1) (1 3 2)) (R2 (0 4 1) (1 4 2)) (R1 (2 3 2))
  (((2 4 1)) (A1 (0 4 1) (1 4 2)) (A2 (2 0 1)))
  (((0 4 1) (1 4 2)) (R1 (2 4 2)) (A2 (0 0 1) (1 0 2))
  (((2 0 1)) (A1 (0 0 1) (1 0 2)))
  (((2 4 2)) (A2 (2 0 2)))
  (((2 3 2)) (R2 (2 4 2)))))
```

Figure 6: Final Controller

References

[1] D. Del Corso and H. Kirrmann and J. D. Nicoud, *Microcomputer Buses and Links*, Academic Press, 1986.

[2] J. Akella, "Performance Modeling and Synthesis of Inter-Process Communication Devices", *Ph.D Thesis, Carnegie Mellon University*, August 1991.

[3] J. Nestor, "Behavioral Synthesis with Interfaces", *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, 112-115, November, 1986.

[4] E. M. Clarke and D. E. Long and K. E. McMillan, "Compositional Model Checking", *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, 353-362, June, 1989.

[5] P. E. Green, "Protocol Conversion", *IEEE Transactions on Communications*, 257-268, March, 1986.

[6] R. Milner, "A Calculus of Communicating Systems", *Lecture Notes on Computer Science*, Springer Verlag, Vol. 92, 1980.

[7] G. Winskel, "A Compositional Proof System on a Category of Labelled Transition Systems", *Computer Science Department, Aarhus University*, No. DAIMI PB - 294, November, 1989.

[8] M. A. Arbib, "Theories of Abstract Automata", *Prentice-Hall*, 1969.

[9] J. L. Peterson, " Petri Net Theory and the Modeling of Systems", *Prentice-Hall Inc*, 1981.

[10] G. Borriello, "Synthesis and Optimization of Interface Transducer Logic", *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, 274-277, November, 1987.