Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 203 (2008) 81–94
www.elsevier.com/locate/entcs

A Model Checking Approach to Protocol Conversion

Roopak Sinha[a,1] Partha S Roop[a,2] Samik Basu[b,3]

[a] *Electrical and Computer Engineering*
University of Auckland
Auckland, New Zealand

[b] *Department of Computer Science*
Iowa State University
Ames, USA

Abstract

System-on-chip verification is an active research area. Of particular interest is protocol conversion, where two components with different protocols are controlled to communicate accurately. We present an approach to protocol conversion using model checking. The temporal logic ACTL is used to describe desired behaviour and finite state machines are used for protocol description. We use tableau-based converter construction and prove that a converter exists only when a successful tableau can be constructed. Liveness is incorporated so that converters satisfy additional constraints on protocol communication. A NuSMV-based implementation has been created and we present results on various problems including a large NuSMV example.

Keywords: model checking, protocol conversion, protocol mismatches

1 Introduction

A System-on-a-chip (SoC) integrates components of a computer system into a single chip with various hardware and software components connected using a central bus such as AMBA [8]. SoC verification is an active area of interest and verification strategies are based on data-flow and/or control-flow analysis of the system. The focus of this paper is *protocol conversion* for *mismatched* protocols [13]. Although *physical connectivity* (interconnection using physical channels) between components can generally be achieved, *logical connectivity*, where processes communicate in the desired fashion, cannot always be guaranteed [13]. A *mismatch* occurs when processes fail to be logically connected. The aim of protocol conversion is to synthesize

[1] Email:rsin077@ec.auckland.ac.nz
[2] Email:p.roop@auckland.ac.nz
[3] Email:sbasu@cs.iastate.edu

doi:10.1016/j.entcs.2008.05.012

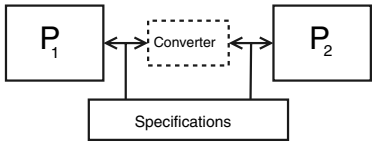

Fig. 1. Protocol conversion

extra glue-logic, called a *converter*, to control mismatched protocols to reconcile mismatches. A converter can control the communication between protocols by employing strategies such as *event hiding* [13], *event translation* [5] and *inhibition* [16]. The automatic generation of a converter is known as *converter synthesis* whereas *convertibility verification* focuses on establishing whether protocols are mismatched and whether a converter exists. Fig. 1 gives an overview of protocol conversion where a converter controls two protocols P_1 and P_2 to satisfy given specifications.

We present a technique using model checking for automatically synthesizing a converter. Protocols, in our setting, are represented using *Kripke Structures* (KS) [7] and the desired properties of the combined protocols are represented using temporal logic `ACTL`, a branching time temporal logic with universal path quantifiers. The logic is particularly interesting and relevant for protocol conversion as mismatches in protocols must be addressed for *every* path of their KS descriptions. Given two KSs P_1 and P_2 and a set of desired properties in `ACTL`, Ψ, the protocol conversion via converter synthesis problem (illustrated in Fig. 1) is equivalent to checking for the existence of a converter under which the protocols satisfy all formulas in Ψ.

Central to our technique is the construction of a tableau where satisfaction of Ψ by the protocols and the converter is defined in terms of the satisfaction of its subformulas (similar to [2]). The tableau construction also results in the synthesis of a converter as the protocol-composition states are explored along with the subformulas of the desired property. The technique leads to local and on-the-fly construction of the converter, one where the state-space of the protocols and the subformulas of the property are explored and expanded as and when needed. In fact, in the event there exists no converter, i.e., the protocols cannot be matched (hard mismatch [10]), our tableau-based technique can potentially identify the failure without exploring the state-space that is irrelevant for failure inference.

The main contributions of this paper are summarized as follows:

- We present a temporal logic based formulation for protocol conversion where temporal logic formulas in `ACTL` are used to specify the desired communication between participating protocols.
- A tableau-based technique for identifying a converter, if one exists, as a glue-logic between composed protocols to reconcile the protocol mismatches and ensure that the desired specifications are satisfied. The tableau is sound and complete and the converter, thus synthesized, is correct by construction.
- The tableau-based technique describes a local and on-the-fly algorithm for converter synthesis—one where the state-space of the protocols being composed are explored only as and when needed to prove or disprove the existence of a con-

verter. The algorithm is polynomial in the size of the participating protocols and the given specifications.

The rest of this paper is organized as follows. We summarize works related to our approach in section 2 and provide a motivating example in section 3. The problem of protocol conversion is described in section 4 and we provide our proposed tableau-based protocol-conversion approach in section 5. Section 7 presents implementation results with concluding remarks in section 8.

2 Related Work

A number of techniques have been developed to address the problem of protocol conversion using a wide range of formal and informal settings with varying degrees of automation—projection approach [13], quotienting [5], conversion seeds [15], synchronization [17], supervisory control theory [11], to name a few. Some techniques, like converters for protocol gateways [3] and interworking networks [4], rely on ad hoc solutions. Some approaches, like protocol conversion based on conversion seeds [15] and protocol projections [13], require significant user expertise and guidance during converter construction. While this problem has been studied in a number of formal settings [10,13,15,17], only recently have some formal verification based solutions been proposed [16,8,11,9].

The closest to our approach are [16,8]. In [16], the authors present an approach towards protocol conversion using finite state machines to represent participating protocols as well as specifications employing a game-theoretic framework to generate a converter. This solution is restricted only to protocols with half-duplex communication between them. D'Silva et al [8] present synchronous protocol automata to allow formal protocol specification and matching, as well as converter synthesis. The matching criteria between protocols are based on whether events are blocking or non-blocking and no additional specifications can be used. The approach allows model checking only as an auxiliary verification step to ensure that conversion is correct.

In contrast to the above techniques, we use temporal logic to represent desired functionality of the combined protocols. Being based on temporal logic, our technique can define desired properties succinctly and with a higher-level of granularity. For example: a desired behavior of the combination may be sequencing of events such that event a in protocol P_1 always happens before event b in P_2. Also, as our technique is based on the (tableau-based) model checking algorithm, the converter synthesized is correct by construction.

The presented approach is similar to the synthesis of discrete controllers with temporal logic and Control-D system [1]. However, the approach in[1] generates controllers that can only perform *disabling*, i.e, transitions in the underlying system can be disabled that lead to the eventual failure of given `CTL` formulas. Additionally, the approach does not handle liveness properties. On the other hand, converters generated using our approach not only perform disabling, but they can also *buffer* events for later use in the communication of the protocols. Additionally, the syn-

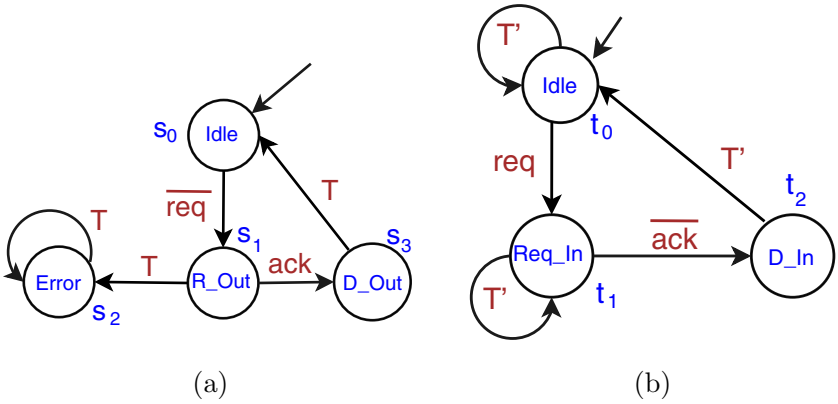

Fig. 2. The producer-consumer protocol pair. (a) Producer P_1, (b) Consumer P_2.

thesized converters can generate *extra control signals* expected as input by one protocol but not emitted by the other, in order to lead the communication between the protocols to states that conform to given specifications.

3 Illustrative Example

We motivate our approach using the following example. Fig. 2 shows the communication protocols of two devices, a producer and a consumer, which need to communicate with each other. In its initial state s_0, the producer protocol P emits a request ($\overline{req}$) and makes a transition to state s_1. In s_1, an acknowledge input (ack) is expected immediately. In case ack is not available, a transition to the error state s_2 is made. In case ack is available, a transition to state s_3 is made where one packet of data is produced (denoted by the D_Out label). From s_3, the producer resets back to its initial state s_0.

The consumer protocol P_2 operates as follows. In its initial state t_0, the consumer awaits a request from the producer protocol. Once a request is received, a transition to state t_1 is made. In state t_1, an acknowledge signal $\overline{ack}$ is emitted and a transition to state t_2 is made. In t_2, a packet of data is read (denoted by the label D_In) and a transition back to the initial state is made. Note that an event a represents an input whereas $\overline{a}$ represents an output. We specify their desired behaviour using the following ACTL formulas:

(i) $\texttt{AG}\neg Error$: The communication never enters a state labelled by $Error$.

(ii) $\texttt{AG}\,[D\_Out \Rightarrow (\ D\_In \ \vee\ \texttt{AXA}(\neg D\_Out\ \texttt{U}\ D\_In)\ )]$: Each data packet emitted by the producer is read by the consumer before another data packet is emitted (no loss).

Given the producer-consumer protocol pair in Fig. 2, it is possible that the unrestricted behavior of the protocols may lead to states that fail to satisfy the above properties. We formalize our solution to resolve these issues in the following sections.

4 Preliminaries

Model of Protocols: Kripke Structures.

Protocols are described using Kripke structures as follows:

Definition 4.1 [Kripke Structure] A Kripke structure (KS) is a finite state machine represented by a tuple $\langle AP,\ S,\ s_0,\ \Sigma,\ R,\ L,\rangle$ where AP is a set of atomic propositions; S is a finite set of states; $s_0 \in S$ is the initial state; Σ is a finite set of events; $R \subseteq S \times \Sigma \times S$ is the transition relation; and $L : S \rightarrow 2^{AP}$ is the state labelling function.

We consider that the transitions in a Kripke structure trigger with respect to a clock. At each clock cycle, the KS checks for the presence of input/output events that can trigger a transition from the current state. If no input/output triggers are present, the transition using the event T (or T') is made. In case there is no T or T'-transition, the protocol remains in the current state. The relations $(s, a, s') \in R$ will be represented by $s \xrightarrow{a} s'$. Given two KS P_1 and P_2 using a shared clock, their combined behavior is given by their *parallel composition* as follows:

Definition 4.2 [Parallel Composition] Given two Kripke structures $P_1 = \langle AP_1,\ S_1,\ s_{0_1},\ \Sigma_1,\ R_1,\ L_1\rangle$ and $P_2 = \langle AP_2,\ S_2,\ s_{0_2},\ \Sigma_2,\ R_2,\ L_2,\rangle$, their parallel composition, denoted by $P_1||P_2$, is $\langle AP_{1||2}, S_{1||2}, s_{0_{1||2}}, \Sigma_{1||2}, R_{1||2}, L_{1||2}\rangle$ where $AP_{1||2} = AP_1 \cup AP_2$; $S_{1||2} = S_1 \times S_2$; $s_{0_{1||2}} = (s_{0_1}, s_{0_2})$; and $\Sigma_{1||2} \subseteq \Sigma_1 \times \Sigma_2$. $R_{1||2} \subseteq S_{1||2} \times \Sigma_{1||2} \times S_{1||2}$ such that

$$(s_1 \xrightarrow{\sigma_1} s_1') \wedge (s_2 \xrightarrow{\sigma_2} s_2') \Rightarrow ((s_1, s_2) \xrightarrow{(\sigma_1,\sigma_2)} (s_1', s_2'))$$

Finally, $L_{1||2}((s_1, s_2)) = L_1(s_1) \cup L_2(s_2)$.

We restrict the scope of this paper to protocols that can be represented as *deterministic* Kripke structures only. A Kripke structure is deterministic if and only if for all states s, the number of outgoing transitions on any event a is less than equal to 1. The parallel composition of P_1 and P_2 in Fig. 2 (assuming a shared clock) is $P_1||P_2$ and is shown in Fig. 3.

Model of Specifications.

ACTL is a branching time temporal logic with universal path quantifiers. It is defined over a set of propositions using temporal and boolean operators as follows:

$$\Psi \rightarrow P \mid \neg P \mid tt \mid ff \mid \Psi \wedge \Psi \mid \Psi \vee \Psi \mid \mathtt{AX}\Psi \mid \mathtt{A}(\Psi\ \mathtt{U}\ \Psi) \mid \mathtt{AG}\Psi$$

Semantics of an ACTL formula, φ denoted by $[\![\varphi]\!]_M$ are given in terms of set of states in a Kripke structure (or a KS), M, which satisfies the formula (see Fig. 4). A state $s \in S$ is said to satisfy a ACTL formula φ, denoted by $M, s \models \varphi$, if $s \in [\![\varphi]\!]_M$. Typically, the context of the semantics, i.e., M in $[\![\ ]\!]_M$ is implicit, and omitted. We also say that $M \models \varphi$ to indicate $M, s_0 \models \varphi$. In this paper, we restrict ourselves to formulas where negations are applied to propositions only.

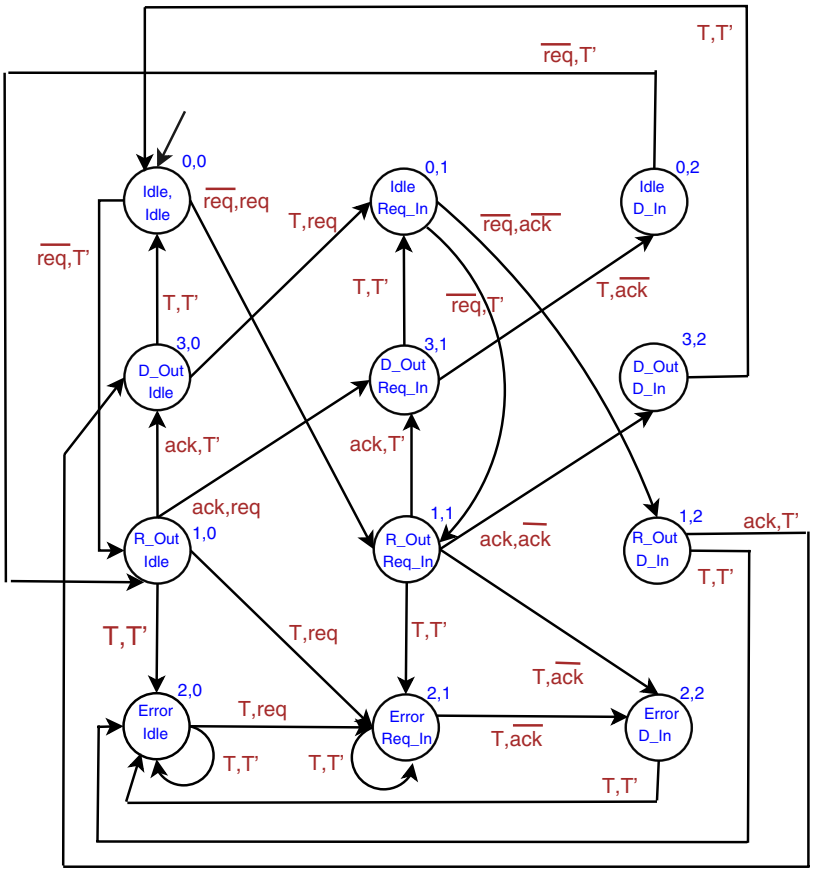

Fig. 3. Unrestricted composition of producer-consumer protocol pair: $P_1||P_2$.

$1 : [\![p]\!] = \{s \mid p \in L(s)\}$ $2 : [\![\neg p]\!] = \{s \mid p \notin L(s)\}$ $3 : [\![tt]\!] = S$ $4 : [\![ff]\!] = \emptyset$

$5 : [\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$ $6 : [\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$

$7 : [\![\mathtt{AX}\varphi]\!] = \{s | \forall s \longrightarrow s' \wedge s' \in [\![\varphi]\!]\}$

$8 : [\![\mathtt{A}(\varphi\ \mathtt{U}\ \psi)]\!] = \{s | \forall s = s_1 \longrightarrow s_2 \longrightarrow \ldots \wedge \exists j.s_j \in [\![\psi]\!]\ \wedge \forall i < j.s_i \in [\![\varphi]\!]\}$

$9 : [\![\mathtt{AG}\varphi]\!] = \{s | \forall s = s_1 \longrightarrow s_2 \longrightarrow \ldots \wedge \forall i.s_i \in [\![\varphi]\!]\}$

Fig. 4. Semantics of ACTL

4.1 Protocol Converters

The composition $P_1||P_2$ (Fig. 3) represents the unconstrained behaviour of the protocols including undesirable paths introduced due to mismatches. A converter is needed to bridge the mismatches appropriately. In this section, we introduce converters and also the control actions of a converter (such as event blocking, event buffering and generation of extra signals) by introducing a new composition of the participating protocols with the converter.

Definition 4.3 [Converter] A converter $\mathcal{C}$ for two protocols P_1 and P_2 is a Kripke structure $\langle AP_{\mathcal{C}}, S_{\mathcal{C}}, s_{\mathcal{C}0}, \Sigma_{\mathcal{C}}, R_{\mathcal{C}}, L_{\mathcal{C}}\rangle$ such that $AP_{\mathcal{C}} = \emptyset$ and $\Sigma_{\mathcal{C}} = (\Sigma_1 \times \Sigma_2) \cup \{(*, *)\}$.

In the above, the event-element $(*,*)$ is a wild-card event tuple, short-hand form of denoting any event-pairs from $\Sigma_1 \cup \Sigma_2$. The composition of a converter with the protocols is performed using the following rule: inputs to (outputs from) a protocol are outputs from (inputs to) the converter, i.e., the participating protocols communicate via the converter which acts as an intermediary. Input and output on the same event are *duals* and we will say that $\mathcal{D}(a,b)$ evaluates to true if $a = \sigma$ $(\bar{\sigma})$ and $b = \bar{\sigma}$ (σ) or if either a and/or b is the wildcard event $*$. We extend $\mathcal{D}$ to operate on pairs of signals, where $\mathcal{D}((a,b),(c,d))$ evaluates to true *iff* both $\mathcal{D}(a,c)$ and $\mathcal{D}(b,d)$ evaluate to true.

After establishing the i/o relationship between a converter and the participating protocols, we now define the control of a converter over the protocols using the // operator as follows.

Definition 4.4 [Lock-Step Converter Composition] Given the KS $P_1||P_2 = \langle AP_{1||2}, S_{1||2}, s_{0_{1||2}}, \Sigma_{1||2}, R_{1||2}, L_{1||2}\rangle$ and a converter $\mathcal{C} = \langle AP_{\mathcal{C}}, S_{\mathcal{C}}, s_{\mathcal{C}0}, \Sigma_{\mathcal{C}}, R_{\mathcal{C}}, L_{\mathcal{C}}\rangle$, the lock-step composition $\mathcal{C}//(P_1||P_2) = \langle AP_{1||2}, S_{\mathcal{C}//(1||2)}, s_{0_{\mathcal{C}//1||2}}, \Sigma_{1||2}, R_{\mathcal{C}//(1||2)}, L_{\mathcal{C}//(1||2)}\rangle$ such that:

(i) $S_{\mathcal{C}//(1||2)} \subseteq S_{\mathcal{C}} \times S_{1||2}$;

(ii) $s_{0_{\mathcal{C}//1||2}} = (s_{0_{\mathcal{C}}}, s_{0_{(1||2)}})$;

(iii) $R_{\mathcal{C}//(1||2)} \subseteq S_{\mathcal{C}//(1||2)} \times \Sigma_{1||2} \times S_{\mathcal{C}//(1||2)}$ where $s_{\mathcal{C}//(1||2)} \overset{(\sigma_1,\sigma_2)}{\longrightarrow} s'_{\mathcal{C}//(1||2)} \in R_{\mathcal{C}//(1||2)}$ when

$$\left[\begin{array}{l} s_{\mathcal{C}} \overset{\sigma_1^c,\sigma_2^c}{\longrightarrow} s'_{\mathcal{C}} \ \wedge \ s_{1||2} \overset{(\sigma_1,\sigma_2)}{\longrightarrow} s'_{1||2} \\ \bigwedge \\ \mathcal{D}(\sigma_1^c,\sigma_1) \ \wedge \ \mathcal{D}(\sigma_2^c,\sigma_2) \end{array}\right] \Rightarrow s_{\mathcal{C}//(1||2)} \overset{(\sigma_1,\sigma_2)}{\longrightarrow} s'_{\mathcal{C}//(1||2)}$$

(iv) $L_{\mathcal{C}//(1||2)}(s_{\mathcal{C}}, s_{1||2}) = L_{1||2}(s_{1||2})$

The transition relation of the protocols composed with a converter ensures that protocols move only when the converter allows that move. As such the lock-step composition // is different from unrestricted composition (Definition 4.2).

5 Tableau-Based Protocol Conversion

Protocol conversion, in addition to reconciling the mismatches, also requires that certain desired behavior is exhibited by the composition of the participating protocols. These desired functionalities are described by a set of ACTL formulas. We will denote this set as Ψ. The converter synthesis problem for protocol conversion is, therefore,

$$\exists \mathcal{C} : \forall \varphi \in \Psi : \ \mathcal{C}//(P_1||P_2) \overset{?}{\models} \varphi$$

I.e. is there a converter $\mathcal{C}$ for P_1 and P_2 such that the given protocols in the presence of $\mathcal{C}$ conforms to all the properties defined by formulas in Ψ?

We present a tableau-based technique for performing protocol conversion using ACTL specifications. This technique has the following advantages:

(i) Local exploration of state-space of the protocols: the protocol transition systems are explored as and when needed to prove or disprove the existence of a converter.

(ii) On-the-fly synthesis of converter: generation of the tableau results in the generation of a converter if such a converter exists.

(iii) Sound and complete: a converter generated using the tableau is correct by construction.

The tableau rules are of the following form:

$$\frac{c//s \models \Psi}{c_1//s_1 \models \Psi_1 \ldots c_n//s_n \models \Psi_n}$$

where s is a state in $P_1||P_2$ and $s_1, s_2, \ldots, s_n$ are a function of s, while $c_1, c_2, \ldots, c_n$ are the states of the converter to be generated. Similarly, Ψ is the set of formulas to be satisfied by s whereas $\Psi_1, \Psi_2, \ldots, \Psi_n$ are some derivatives of Ψ. The numerator represents the obligation to be satisfied, i.e., s in the presence of a converter state c must satisfy the set of formulas in Ψ and in order to realize that, each obligation in the denominator must be fulfilled.

The tableau is initiated by a tableau-node resulting from the composition of the start state of the unrestricted composition of P_1 and P_2 and a generated start state c_0 of a possible converter. The construction proceeds by matching the current tableau-node with the numerator of a tableau rule and obtaining the denominator which constitutes the next set of tableau-nodes. Fig. 5 presents our tableau-rules for converter synthesis and protocol conversion.

The rule emp corresponds to the case when there is no obligation to be satisfied by the composition; any converter is possible in this case, i.e., the converter allows all possible behavior of the protocol composition at state s.

The prop rule states that a converter is synthesizable only when the obligation of satisfying the proposition is released by the protocol composition state s; otherwise there exists no converter. Once the propositional obligation is met, the subsequent obligation is to satisfy the rest of the formulas in the set Ψ.

The $\wedge$-rule states that the satisfaction of the conjunctive formula depends on the satisfaction of each of the conjuncts. The $\vee$-rules are the duals of $\wedge$-rule. The Rule $\mathtt{unr}_{au}$ depends on the semantics of the temporal operator AU. A state is said to satisfy $\mathtt{A}(\varphi\ \mathtt{U}\ \psi)$ if and only if it either satisfies ψ or satisfies φ and evolves to new states each of which satisfies $\mathtt{A}(\varphi\ \mathtt{U}\ \psi)$. These equivalences can be directly derived from the semantics of AU formulas. Similarly, $\mathtt{AG}\varphi$ is satisfied by states which satisfy φ and whose all next states satisfy $\mathtt{AG}\varphi$ (Rule $\mathtt{unr}_{ag}$).

Finally, $\mathtt{unr}_s$ is applied when the formula set in the numerator Ψ consists formulas of the form $\mathtt{AX}\varphi$. Satisfaction of these formulas demands that all next states

$$\text{emp}\ \frac{c//s \models \{\}}{\bullet} \qquad \text{prop}\ \frac{c//s \models [\{p\} \cup \Psi]}{c//s \models \Psi}\ p \in L(s) \vee \models p$$

$$\wedge\ \frac{c//s \models [\{\varphi_1 \wedge \varphi_2\} \cup \Psi]}{c//s \models [\{\varphi_1, \varphi_2\} \cup \Psi]}$$

$$\vee_1\ \frac{c//s \models [\{\varphi_1 \vee \varphi_2\} \cup \Psi]}{c//s \models [\{\varphi_1\} \cup \Psi]} \qquad \vee_2\ \frac{c//s \models [\{\varphi_1 \vee \varphi_2\} \cup \Psi]}{c//s \models [\{\varphi_2\} \cup \Psi]}$$

$$\mathtt{unr}au\ \frac{c//s \models [\{\mathtt{A}(\varphi\ \mathtt{U}\ \psi)\} \cup \Psi]}{c//s \models [(\psi \vee (\varphi \wedge \mathtt{AXA}(\varphi\ \mathtt{U}\ \psi))) \cup \Psi]}$$

$$\mathtt{unr}ag\ \frac{c//s \models [\{\mathtt{AG}\varphi\} \cup \Psi]}{c//s \models [(\varphi \wedge \mathtt{AXAG}\varphi) \cup \Psi]}$$

$$\mathtt{unr}_S\ \frac{c//s \models \Psi}{\exists \pi \subseteq \Pi.\ (\forall \sigma \in \pi.\ c_\sigma//s_\sigma \models \Psi_{AX})} \quad \begin{cases} \Psi_{\mathtt{AX}} = \{\varphi_k \mid \mathtt{AX}\varphi_k \in \Psi\} \\ \Pi = \{\sigma \mid (s) \xrightarrow{\sigma} (s_\sigma)\} \\ c_\sigma = c' : c \xrightarrow{\sigma'} c' \wedge \mathcal{D}(\sigma, \sigma') \end{cases}$$

Fig. 5. Tableau Rules for converter generation

of the $c//s$ must satisfy every φ where $\mathtt{AX}\varphi \in \Psi$, i.e., $c//s$ satisfies all elements of Ψ_{AX}.

Note that unrestricted behavior of the protocol (where c allows all the transitions from s) may not be able to satisfy this obligation; however, a converter can be generated such that c allows a *subset* (π) of all possible transitions (Π) from s and these transitions lead to states which satisfy the formulas in Ψ_{AX} (as stated by the unr_s rule). If there are k outgoing transitions from s, there are 2^k choices; however, the tableau considers k choices (one for each successor) and unr_s leads to k possible denominators–one denominator per transition from s. These choices can then be aggregated to represent all enabled transitions of s. Any denominators that return failure result in the corresponding successors of s being disabled by the converter [4].

Finitizing the tableau.

It is important to note that the resulting tableau can be of infinite depth as each recursive formula expression AU or AG can be unfolded infinitely many times.

This problem arising due to unbounded unfolding of the formula expressions can be addressed using the fixed point semantics of the formulas $\mathtt{AG}\varphi$ and $\mathtt{A}(\varphi\ \mathtt{U}\ \psi)$. The

[4] However, instead of examining all possible subsets, it is sufficient for the converter state c to allow just one transition from s such that $c'//s'$ satisfies all formulas in Ψ_{AX}, although such a converter may be too restrictive.

former is a greatest fixed point formula while the later is a least fixed point formula.

$$\texttt{AG}\varphi \equiv Z_{\texttt{AG}} =_{\nu} \varphi \wedge AXZ_{\texttt{AG}},$$
$$\texttt{A}(\varphi\ \texttt{U}\ \psi) \equiv Z_{\texttt{AU}} =_{\mu} \psi \vee (\varphi \wedge AXZ_{\texttt{AU}})$$

The greatest (least) solution for $Z_{\texttt{AG}}$ ($Z_{\texttt{AU}}$) is the semantics of $\texttt{AG}(\varphi)$. It can be shown (details are omitted) that satisfaction of the greatest fixed point formula is realized via loops in the model; while satisfaction of the least fixed point formula demands the existence of a loop-free tableau. As such, if a tableau-node $c'//s \models \Psi$ is visited and there exists a prior node $c//s \models \Psi$ i.e. the same tuple s paired with the same Ψ is seen in a tableau path, we verify whether there exists a least fixed point formula AU in Ψ; if such a formula is present, we say that the tableau path resulted in an unsuccessful path; otherwise, we terminate the tableau path successfully and equate c' with c (a loop in the converter is generated).

Complexity.

The tableau considers all possible subformulas of the given set of desired properties. Each such subformula is paired with all possible states in the protocol-pair. The complexity of the tableau construction is $O(|S| \times 2^{|\varphi|})$ where S is the number of states in the protocol pairs and $|\varphi|$ is the size of the formula expressing the desired properties (the conjunction of all properties).

The following theorem follows from the above discussion.

Theorem 5.1 (Sound and Complete) *Two protocols P_1 and P_2 are compatible* wrt *to a set Ψ of ACTL formulas ($\forall\varphi \in \Psi : \mathcal{C}//(P_1||P_2) \models \varphi$) if and only if there exists a successful tableau for the tableau node $c_0//s_0 \models \Psi$ where s_0 is the start state of $P_1||P_2$ and c_0 is the start state of $\mathcal{C}$.*

6 Live Converters

For two protocols P_1 and P_2 and a set of ACTL specifications Ψ, the tableau-based approach formulated above can generate multiple converters. This is because the rules $\vee$ and unr_s may lead to several choices for constructing the tableau-node denominator. Some of the generated converters, therefore, may disable protocol-behavior and lead to conformance of the desired property vacuously. For example, properties of the form $\phi \Rightarrow \psi$ will be satisfied by the converted protocol pairs if ϕ is not satisfied.

To counter this situation, we can impose further restrictions on converter generation by including *liveness* conditions that need to be satisfied by the resulting system $\mathcal{C}//P_1||P_2$. Such liveness conditions can be defined using ACTL and used as input to tableau along with desired properties. The goal will be avoid construction of converters that will lead to violation of liveness properties by the converted protocols.

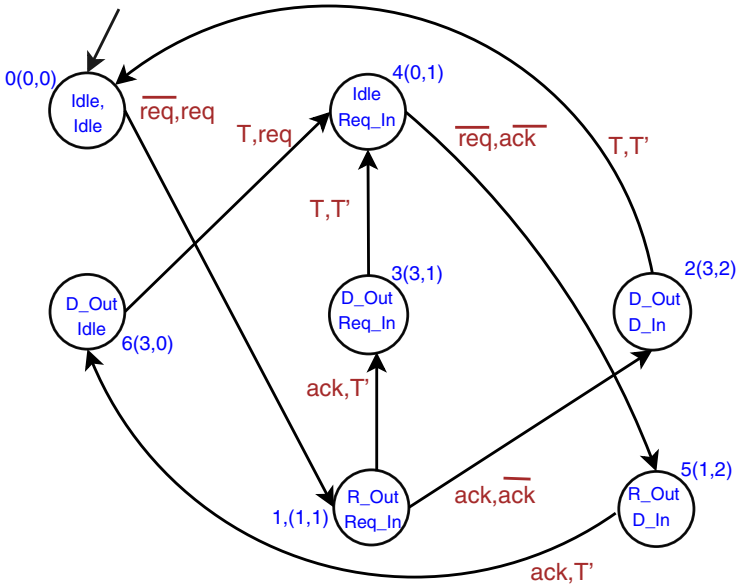

Fig. 6. The combined system $\mathcal{C}//P_1||P_2$ (Figures 2 and 3).

For the producer-consumer example, we use the following liveness conditions:

- $\texttt{AGA}(true\ \texttt{U}\ D\_In)$, $\texttt{AGA}(true\ \texttt{U}\ D\_Out)$: $\mathcal{C}$ must allow the producer to always eventually write data and the consumer to always eventually consumer some data.
- $\texttt{AG}[D\_Out \Rightarrow (\ D\_In \vee \texttt{AXA}(\neg R\_Out\ \texttt{U}\ D\_In)\ )]$: Once data is written, no further requests are allowed before a read operation is performed.

The combined system $\mathcal{C}//(P_1||P_2)$ is shown in Fig. 6. The converter $\mathcal{C}$ obtained for the producer-consumer example is a maximally permissive converter that ensures that $\mathcal{C}//(P_1||P_2)$ satisfies the above liveness constraints. For better readability in Fig. 6, we have annotated each state with $i(j,k)$ where i denotes the state of the generated converter while j and k are states of P_1 (producer) and P_2 (consumer) respectively.

7 Results

A protocol conversion tool employing the tableau construction approach has been implemented by extending the NuSMV model checker [6]. The implementation takes as input the Kripke structure representation of two protocols P_1 and P_2 (obtained from NuSMV models) and a set Ψ of ACTL properties from the user. It proceeds by computing the parallel composition $P_1||P_2$ and then uses the tableau rules to realize the converter, if it exists. The results table (Tab.1) contains four columns. The first two columns contain the description and size (number of states) of the participating protocols. The ACTL properties used are shown in the third column with the size of the converter shown in column 4. The first five problems are well-known protocol conversion problems [16,13]. The next problem is a producer-

$P_1(\|S_{P_1}\|)$	$P_1(\|S_{P_2}\|)$	ACTL Properties	$\mathcal{C}(\|S_{\mathcal{C}}\|)$
Master (3)	Slave (3)	$\mathtt{AG}(\neg Req\_InUR\_Out)$, $\mathtt{AG}[R\_Out \Rightarrow ((Req\_In) \vee \mathtt{AXA}(\neg R\_Out\ \mathtt{U}\ Req\_In))]$, $\mathtt{A}(\neg G\_Out\ \mathtt{U}\ R\_Out)$, $\mathtt{A}(\neg Gnt\_In\ \mathtt{U}\ G\_Out)$	6
ABP sender(6)	NP receiver(4)	$\mathtt{AGA}(\neg A\_Out\ \mathtt{U}\ ACC)$, $\mathtt{AG}[A\_Out \Rightarrow (ACC \vee \mathtt{AXA}(\neg A\_Out\ \mathtt{U}\ ACC))]$	8
ABP receiver(8)	NP sender(3)	$\mathtt{AGA}(\neg A\_Out\ \mathtt{U}\ ACC)$, $\mathtt{AG}[(A+ \Rightarrow (ACC \vee \mathtt{AXA}(\neg A\_Out\ \mathtt{U}\ ACC))]$	8
Poll-End Receiver(2)	Ack-Nack Sender(3)	$\mathtt{AG}[Data\_Out \Rightarrow (Data\_In \vee \mathtt{AXA}(Data\_In\ \mathtt{U}\ Data\_Out))]$	6
Handshake (2)	Serial(2)	$\mathtt{AGA}(\neg A\ \mathtt{U}\ A')$, $\mathtt{AGA}(\neg B\ \mathtt{U}\ B')$, $\mathtt{AG}(A' \Rightarrow \mathtt{AXA}(\neg A'\ \mathtt{U}\ A))$	3
Multi-write master protocol(3) 8-bit Write	Single-read slave protocol(4) 8-bit Read	$\mathtt{AG}(\neg Error)$,$\mathtt{A}(\neg D\_Out\ \mathtt{U}\ Req\_In)$ $\mathtt{A}(\neg Req\_In\ \mathtt{U}\ R\_Out)$	8
Mutex Process 1 (3)	Mutex Process 2 (3)	$\mathtt{AG}(\neg critical1 \vee \neg critical2)$	7
MCP missionaries	MCP cannibals (30)	$\mathtt{AGAF}((MCP.missionaries = 0) \wedge (MCP.cannibals = 0))$	22
4-bit ABP Sender	Modified Receiver (166432)	$\mathtt{AGAF} sender.state = get$	14312

Table 1
Implementation Results

consumer example where the producer can produce multiple 8-bit data after each handshake whereas the slave can only read one 8-bit data after each handshake. The generated converter controls the communication between the two components such that paths where data is lost are never reached. The final three results are well-known NuSMV examples modified to create a mismatch. Note that size entry in the second column for the final two results refers to the combined size of the system (size of $P_1||P_2$) for these examples.

8 Conclusions and Future Directions

Protocol conversion to resolve protocol mismatches is an active research area. A number of solutions have been proposed. Some approaches require significant user input and guidance, while some only partly address the protocol conversion problem. Most formal approaches work on protocols that have unidirectional communication and use finite state machines to describe specifications. In this paper we propose a formal approach to protocol conversion which alleviates the above problems. Specifications are described in temporal logic and protocols are allowed to be bidirectional. A tableau-based approach using the model checking framework is used to generate converters in polynomial time. We prove that the approach is sound and complete and provide implementation results.

The presented approach uses ACTL to describe desired specifications. The extension to the more expressive logic CTL requires minimal effort but the presence of existential formulas in CTL will increase the complexity to $EXPTIME$-complete as protocol conversion under CTL is equivalent to *module checking* [12,1] problem. Similarly, tableau rules for LTL will result in $PSPACE$ complexity of protocol conversion. Future work includes the unification of various protocol conversion issues under the presented framework. The technique can be extended to resolve data-width mismatches [8], clock-mismatches [14] and interface-mismatches between protocols. Data-width mismatches occur when protocols have varying word-sizes. A converter must therefore ensure that no data is lost during inter-protocol communication. Clock-mismatches occur when protocols operate using clocks that may be running at different frequencies. Interface mismatches occur when protocols use inconsistent naming conventions for control signals, thus requiring the converter to perform event translation [5]. Another issue is the handling of *uncontrollable* actions [11,1]. Some transitions in $P_1||P_2$ may be uncontrollable and therefore cannot be disabled. An extension to the presented tableau-based converter generation approach to generate a converter, if possible, under these additional restrictions is endeavored.

References

[1] M. Antoniotti. *Synthesis and verification of discrete controllers for robotics and manufacturing devices with temporal logic and the Control-D system*. PhD thesis, New York University, New York, 1995.

[2] Girish Bhat, Rance Cleaveland, and Orna Grumberg. Efficient on-the-fly model checking for CTL*. In *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 388–397, June 1995.

[3] G V Bochmann. Deriving protocol converters for communication gateways. *IEEE Transactions on Communications*, 38(9):1298–1300, September 1990.

[4] F M Burg and N D Iorio. Networking of networks: Interworking according to osi. *IEEE Journal on Selected Areas in Communications*, 7(7):1131–1142, September 1989.

[5] Kenneth L Calvert and Simon S Lam. Formal methods for protocol conversion. *IEEE Journal on Selected Areas in Communication*, 8(1):127–142, 1990.

[6] R. Cavada, Alessandro Cimatt, E. Olivetti, M. Pistore, and M. Roveri. *NuSMV 2.1 User Manual*, June 2003.

[7] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[8] Vijay D'Silva, S Ramesh, and Arcot Sowmya. Synchronous protocol automata : A framework for modelling and verification of soc communication architectures. In *DATE*, pages 390–395, 2004.

[9] Saurav Gorai, Saptarshi Biswas, Lovleen Bhatia, Praveen Tiwari, and Raj S. Mitra. Session 42: simulation assisted formal verification: Directed-simulation assisted formal verification of serial protocol and bridge. In *Proceedings of the 43rd annual conference on Design automation DAC '06*, pages 731 – 736, 2006.

[10] P. Green. Protocol conversion. *IEEE Transactions on Communications*, 34(3):257–268, March 1986.

[11] R. Kumar and S. S. Nelvagal. Protocol conversion using supervisory control techniques. In *IEEE International Symposium on Computer-Aided Control System Design*, pages 32–37, 1996.

[12] O. Kupferman, M. Y. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164:322–344, 2001.

[13] S Lam. Protocol conversion. *IEEE Transactions on Software Engineering*, 14(3):353–362, 1988.

[14] J Lefebvre. *Esterel v7 Reference Manual-Initial Standardization Proposal*, 2005.

[15] K. Okumura. A formal protocol conversion method. In *ACM SIGCOMM 86 Symposium*, pages 30–37, 1986.

[16] R. Passerone, L. de Alfaro, T. A. Henzinger, and A. L. Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: Two faces of the same coin. In *International Conference on Computer Aided Design ICCAD*, 2002.

[17] J. C. Shu and Ming T. Liu. A synchronization model for protocol conversion. *Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies. Technology: Emerging or Converging? INFOCOM '89*, pages 276–284, 1989.