Linking Theorem Proving and Model-Checking with Well-Founded Bisimulation *

Panagiotis Manolios[1], Kedar Namjoshi[2], and Robert Sumners[3]

[1] Department of Computer Sciences, University of Texas at Austin
pete@cs.utexas.edu
[2] Bell Laboratories, Lucent Technologies
kedar@research.bell-labs.com
[3] Department of Electrical and Computer Engineering
University of Texas at Austin
sumners@cerc.utexas.edu

Abstract. We present an approach to verification that combines the strengths of model-checking and theorem proving. We use theorem proving to show a bisimulation up to stuttering on a—potentially infinite-state—system. Our characterization of stuttering bisimulation allows us to do such proofs by reasoning only about single steps of the system. We present an on-the-fly method that extracts the reachable quotient structure induced by the bisimulation, if the structure is finite. If our specification is a temporal logic formula, we model-check the quotient structure. If our specification is a simpler system, we use an equivalence checker to show that the quotient structure is stuttering bisimilar to the simpler system. The results obtained on the quotient structure lift to the original system, because the quotient, by construction, is refined by the original system.

We demonstrate our methodology by verifying the alternating bit protocol. This protocol cannot be directly model-checked because it has an infinite-state space; however, using the theorem prover ACL2, we show that the protocol is stuttering bisimilar to a small finite-state system, which we model-check. We also show that the alternating bit protocol is a refinement of a non-lossy system.

1 Introduction

We propose an approach to verification that combines the strengths of the model-checking [CE81,QS82,CES86] and the automated theorem proving (*e.g.*, [BM79,GM93]) approaches. We use a theorem prover to reduce an infinite-state (or large finite-state) system to a finite-state system, which we then handle using automatic methods.

The reduction amounts to proving a stuttering bisimulation [BCG88] that preserves properties of interest. Two states are stuttering bisimilar if they are equivalent up to next-time free CTL^* properties ($CTL^*\backslash X$). $CTL^*\backslash X$ can be used to state most properties of asynchronous systems (including fairness) and many timing-independent properties of synchronous hardware. Bisimulation—the usual notion of branching-time equivalence—is not appropriate when comparing systems at different levels of abstraction because a single step of the abstract system may correspond to many steps of the concrete system. Weak bisimulation [Mil90] allows such comparisons, but does not preserve $CTL^*\backslash X$ properties. We introduce well-founded equivalence bisimulation (WEB), a characterization of stuttering bisimulation

* Manolios's research was supported in part by the Admiral B.R. Inman Centennial Chair in Computing Theory. Namjoshi's research was supported in part by NSF Grant CCR-9415496 and SRC Grant 97-DP-388. Sumners's research was supported in part by an AMD/SRC fellowship.

N. Halbwachs and D. Peled (Eds.): CAV'99, LNCS 1633, pp. 369–379, 1999.

that is based on well-founded bisimulation [Nam97]. A proof that a relation is a WEB involves checking that each action of the program preserves the relation. Such single step proofs can be checked by theorem provers more readily than proofs based on the original definition of stuttering bisimulation.

A WEB induces a quotient structure that is equivalent (up to stuttering) with the original system. The idea is to check the quotient structure, but constructing the quotient structure can be difficult because determining if there is a transition between states in the quotient structure depends on whether there is a transition between some pair of related states in the original system (the number of such pairs may be infinite). Moreover, the quotient structure may be infinite-state, but the set of its reachable states may be finite. To address these two concerns, we introduce an on-the-fly algorithm that for a large class of systems automatically extracts the quotient structure. Once the quotient structure is extracted, we can model-check it or we can use a WEB equivalence checker to compare it with another system.

We are interested in *mechanical verification*; by this we mean that every step in the proof of correctness (except for meta-theory and mechanical tools) is checked mechanically. The theorem prover we use is ACL2 [KM97]. ACL2 is an extended version of the Boyer-Moore theorem prover [BM79]. ACL2 is based on a first-order, executable logic of total recursive functions with induction. We have implemented a μ-calculus model checker with Büchi automata, a WEB equivalence checker, and the quotient extraction algorithm in ACL2; this allows us to perform all of the verification in ACL2 (this is possible because ACL2 is executable). The ACL2 files used are available upon request from the first author.

We demonstrate our approach by verifying the alternating bit protocol [BSW69]. We chose the alternating bit protocol because it has been used as a benchmark for verification efforts, and since this is the first paper to use WEBs for verifying systems, it makes sense to compare our results with existing work. The alternating bit protocol has a simple description but lengthy hand proofs of correctness (*e.g.*, [BG94]), it is infinite-state, and its specification involves a complex fairness property. We have found it to be surprisingly difficult to verify mechanically; many previous papers verify various versions of the protocol (*e.g.*, [Mil90,CE81,HS96,BG96,MN95]), but all make simplifying assumptions, either by restricting channels to be bounded buffers, by ignoring data, or by ignoring fairness issues.

In the next section, we discuss notation and present the theoretical background, including the definitions of WEB, quotient structure, and refinement; related theorems are also presented. Due to space limitations, proofs of the theorems are omitted; they will appear in a future paper. We assume that the reader is familiar with the temporal logic CTL^* [EH86]. In Section 3, we present the ACL2 formalization of the alternating bit protocol. In Section 4, we present the proof of correctness and in Section 5, we present concluding remarks and comparisons to other work.

2 Theoretical Background

2.1 Preliminaries

$\mathbb{N}$ denotes the natural numbers, *i.e.*, $\{0, 1, \dots\}$. Function application is denoted by an infix dot "." and is right associative. $\langle Qx : r : b \rangle$ denotes a quantified expression, where Q is the quantifier, x the dummy, r the range of x (true if omitted), and b the body. "Such that" and "with respect to" are abbreviated by "s.t." and "w.r.t.", respectively. The cardinality of a set S is denoted by $|S|$. For a relation R, we write sRw instead of $\langle s, w \rangle \in R$. We write $R(S)$ for the image of S under R, *i.e.*, $R(S) = \{y : \langle \exists x : x \in S : xRy \rangle\}$ and $R|_A$ for R left-restricted to the set A, *i.e.*, $R|_A = \{\langle a, b \rangle : (aRb) \ \wedge \ (a \in A)\}$. A *well-founded structure* is a pair $\langle W, \prec \rangle$ where W is a set and $\prec$ is a binary relation on W s.t. there are no infinitely decreasing sequences on W w.r.t. $\prec$. We abbreviate $((s \prec w) \quad \vee \quad (s = w))$ by $s \preccurlyeq w$. From highest to lowest binding power, we have: parentheses, function application, binary relations (*e.g.*, sBw), equality ($=$) and membership ($\in$), conjunction ($\wedge$) and disjunction ($\vee$), implication ($\Rightarrow$), and finally, binary equivalence ($\equiv$). Spacing is used to reinforce binding: more space indicates lower binding.

Definition 1 *(Transition System)*
A Transition System (TS) is a structure $\langle S, \dashrightarrow, L, I, AP \rangle$, where S is a non-empty set of states, $\dashrightarrow \ \subseteq S \times S$ is the *transition relation* (which must be left total), AP is the set of *atomic propositions*, $L : S \rightarrow 2^{AP}$ is the *labeling function* which maps each state to the subset of atomic propositions that hold at that state, and I is the (non-empty) set of *initial states*. We only consider transition systems with countable branching.

Definition 2 *(Well-Founded Equivalence Bisimulation (WEB))*
B is a well-founded equivalence bisimulation on TS $M = \langle S, \dashrightarrow, L, I, AP \rangle$ iff:

1. B is an equivalence relation on S; and
2. $\langle \forall s, w \in S : sBw : L.s = L.w \rangle$; and
3. There exists a function, $rank : S \times S \rightarrow W$, s.t. $\langle W, \prec \rangle$ is well-founded, and

$$\begin{array}{l} \langle \forall s, u, w \in S : sBw \quad \wedge \quad s \dashrightarrow u : \\ \qquad \langle \exists v : w \dashrightarrow v : uBv \rangle \quad \vee \\ \qquad (uBw \quad \wedge \quad rank.(u,u) \prec rank.(s,s)) \quad \vee \\ \qquad \langle \exists v : w \dashrightarrow v : sBv \quad \wedge \quad rank.(u,v) \prec rank.(u,w) \rangle \rangle \end{array}$$

We will call a pair $\langle rank, \langle W, \prec \rangle \rangle$ satisfying condition 3 in the above definition, a *well-founded witness*. Note that to prove a relation is a WEB, reasoning about single steps of $\dashrightarrow$ suffices, whereas, to prove a stuttering bisimulation, one has to reason about infinite paths (the definition of stuttering bisimulation [BCG88] is essentially the same as the above definition except for 3, which states that for any s, w s.t. sBw, any infinite path from s can be "matched" by an infinite path from w.). It is much simpler to use a theorem prover to reason about single steps of $\dashrightarrow$ than it is to reason about infinite paths; this is the motivation for the above definition.

Theorem 1 (cf. [BCG88,Nam97]) *If B is a WEB on TS M and sBw, then for any CTL$^*\backslash$X formula f, $M, s \models f$ iff $M, w \models f$.*

For an equivalence relation B on TS M, a *quotient structure* M/B (read M "mod" B) can be defined, whose states are the equivalence classes of B and whose transition relation is derived from the transition relation of M. Quotient structures can be much smaller than the original: an equivalence relation with finitely many classes induces a finite quotient structure (of a possibly infinite-state system).

Definition 3 *(Quotient Structure)*
Let $M = \langle S, \dashrightarrow, L, I, AP \rangle$ be a TS and let B be a WEB on M. The class of state s is denoted by $[s]$. The quotient structure M/B is the TS $\langle \mathcal{S}, \rightsquigarrow, \mathcal{L}, \mathcal{I}, AP \rangle$, where:

1. $\mathcal{S} = \{[s] \ : \ s \in S\}$; and
2. $\mathcal{L}.C = L.s$, for some s in C (equivalent states have the same label); and
3. $\mathcal{I} = \{[s] \ : \ s \in I\}$; and
4. The transition relation is given by: For $C, D \in \mathcal{S}$, $C \rightsquigarrow D$ iff either

a) $C \neq D$ and $\langle \exists s, w : s \in C \wedge w \in D : s \dashrightarrow w \rangle$, or
b) $C = D$ and $\langle \forall s : s \in C : \langle \exists w : w \in C : s \dashrightarrow w \rangle \rangle$

(The case distinction is needed to prevent spurious self loops in the quotient structure, arising from stuttering steps in the original structure.)

Theorem 2 (cf. [Nam97]) *If B is a WEB on TS M, then there is a WEB on the union of M and M/B that relates states from M with their equivalence classes.*

Corollary 1 *For any $CTL^{*}\backslash X$ formula f, $M, s \models f$ iff $M/B, [s] \models f$.*

2.2 Quotient Extraction

We define a class of functions which we call "representative" functions. As we will see, representative functions allow us to extract finite quotient structures automatically.

Definition 4 *(Representative Function)*
Let $M = \langle S, \dashrightarrow, L, I, AP \rangle$ be a TS and let B be a WEB on M, with well-founded witness $\langle rank, \langle W, \prec \rangle \rangle$. Let $rep : S \rightarrow S$; then rep is a representative function for M w.r.t. B if for all $s, w \in S$:

1. $sBw \quad \equiv \quad rep.s = rep.w$; and
2. $rep.rep.s = rep.s$; and
3. $rank.(w, rep.s) \preccurlyeq rank.(w, s)$; and
4. $rank.(rep.s, rep.s) \preccurlyeq rank.(s, s)$

Theorem 3 *Let rep be a representative function for TS $M = \langle S, \dashrightarrow, L, I, AP \rangle$ w.r.t. WEB B. Let $S' = rep(S)$, and let $M' = \langle S', \rightrightarrows, L|_{S'}, rep(I), AP \rangle$, where $s \rightrightarrows u$ iff $\langle \exists v : s \dashrightarrow v : rep.v = u \rangle$. Then M' is M/B, up to a renaming of states.*

Representative functions are very useful (when they exist) because they identify states that have all of the branching behavior of their class. They allow one to view the quotient structure as a submodel of the original structure, and they are used in the following on-the-fly algorithm for constructing quotient structures.

Algorithm 1 *Quotient Construction*
Given a representative function, rep, for $M = \langle S, \dashrightarrow, L, I, AP \rangle$ w.r.t. B, one can construct the reachable quotient structure induced by B if $rep(I)$ is finite and computable, and if for all $s \in S, rep(\dashrightarrow (s))$ is finite and computable. We start by mapping I to $rep(I)$ and then explore the state space, *e.g.*, by a breadth first traversal. Given a state, s, in the induced quotient structure (recall that s is also a state in the original structure), we compute the set $rep(\dashrightarrow (s))$, which is the set of next states of s in the quotient structure. This process is repeated until no new states are generated. If the set of reachable quotient structure states is finite, the process will terminate.

2.3 Refinement

In this section, $M = \langle S, \dashrightarrow, L, I, AP \rangle$ and $M' = \langle S', \dashrightarrow', L', I', AP' \rangle$. M and M' are *isomorphic* if there is a bijection $f : S \rightarrow S'$ s.t. $s \dashrightarrow w$ iff $f.s \dashrightarrow' f.w$, and $f(I) = I'$. M and M' are *β-isomorphic* if they are isomorphic, β is a subset of both AP and AP', and L

and L' agree when restricted to β, *i.e.*, for any $p \in \beta, p \in L.s$ iff $p \in L'.f.s$ for all s. We say M and M' are WEB if $AP = AP'$ and there are WEBs on M and M' s.t. the quotient structures induced are AP-isomorphic. M and M' are β-WEB if β is a subset of both AP and AP' and the structures obtained from M and M' by restricting L and L' to β are WEB. If M and M' are AP'-WEB, then we say that M is a *refinement* of M'.

Theorem 4 *(Refinement)*

1. *If M is a refinement of M', then any $CTL^*\backslash X$ formula that holds in M' holds in M.*
2. *If M and M'' are β-isomorphic, M'' is a refinement of M', and AP' is a subset of β, M is a refinement of M'.*

Note that the converse of the first part of the theorem does not hold because AP may be a proper superset of AP'. Refinement in a branching-time framework corresponds to refining atomicity in such a way that when the variables introduced for the refinement are hidden, the resulting system and the original system are WEB. Refinement depends crucially on stuttering [Lam80] because we are comparing systems at differing levels of abstraction and any reasonable correctness condition will not make assumptions about how long it takes for something to happen, *i.e.*, the condition should be stuttering insensitive (*i.e.*, the condition will not use X, the next-time temporal operator).

3 Protocol

The alternating bit protocol is used to implement reliable communication over faulty channels. We present the protocol from the view of the sender and receiver first and then in complete detail. The sender interacts with the communication system via the register *smsg* and the flag *svalid*. The sender can assign a message to *smsg* provided it is invalid, *i.e.*, *svalid* is false. The receiver interacts with the communication system via the register *rmsg* and the flag *rvalid*. The receiver can read *rmsg* provided it is valid, *i.e.*, *rvalid* is not false; when read, *rmsg* is invalidated. Figure 1 depicts the protocol from this point of view.

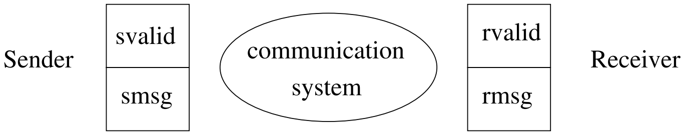

Fig. 1. Protocol from sender's and receiver's view

The communication system consists of the flags *sflag* and *rflag* as well as the two lossy, unbounded, and FIFO channels *s2r* and *r2s*. The idea behind the protocol is that the contents of *smsg* are sent across *s2r* until an acknowledgment for the message is received on *r2s*, at which point a new message can be transmitted. Similarly, acknowledgments for a received message are sent across *r2s* until a new message is received. In order for the receiving end to distinguish between copies of the same message and copies of different messages, each message is tagged with *sflag* before being placed on *s2r*. When a new message is received, *rflag* is assigned the value of the message tag and gets sent across *r2s*; this also allows the sending end to distinguish acknowledgments. There may be an arbitrary number of copies of a message (or an acknowledgment) on the channels, and it turns out that there are at

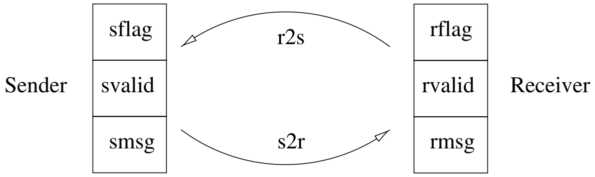

Fig. 2. Alternating Bit Protocol

most two distinct messages (or acknowledgments) on the channels, hence binary flags suffice. Figure 2 depicts the protocol.

The above discussion is informal; a formal description follows, but first we discuss notation. We have formalized the protocol and its proof in ACL2, however, for presentation purposes we describe the formalization using standard notation. We remain faithful to the ACL2 formalization, *e.g.*, we do not use types: functions that appear typed are really underspecified, but total. The concatenation operator on sequences is denoted by ":", but sometimes we use juxtaposition; "ϵ" denotes the empty sequence; *head.s* is the first element of sequence s; *tail.s* is the sequence resulting from removing the first element from s; $|s|$ is the size of the sequence. Messages are pairs; *info* returns the first component of a message and *flag* returns the second.

A state is an eight-tuple $\langle sflag, svalid, smsg, s2r, r2s, rflag, rvalid, rmsg \rangle$; *state* is a predicate that recognizes states. The *sflag* of state s is denoted *sflag.s* and similarly for the other fields. Rules are functions from states into states; they are listed in Table 1 and are of the form $G \rightarrow A$; if A is used as a rule, it abbreviates $\mathsf{true} \rightarrow A$. Rule $G \rightarrow A$ defines the function $(\lambda s : \underline{\mathsf{if}}\ G.s\ \underline{\mathsf{then}}\ A.s\ \underline{\mathsf{else}}\ s)$. We now define the transition relation, R (corresponding to $\dashrightarrow$ in the previous section): sRw iff s is a state and w can be obtained by applying some rule to s.

We have defined the states and transition relation of the alternating bit protocol. The states are labeled with an eight-tuple, as mentioned above. It should be clear that we can convert this type of labeling into a labeling over atomic propositions (boolean variables) by introducing enough—in this case an infinite number of—atomic propositions, therefore, the alternating bit protocol defines a TS, ABP.

4 Protocol Verification

We give an overview of the verification of the alternating bit protocol. ABP'' is the alternating bit protocol, with some variables distorted. Let β be the set of variables that are not distorted; then ABP and ABP'' are β-isomorphic. We define a relation B and prove that B is a WEB on ABP''. We define *rep*, a representative function on ABP'' w.r.t. B. We use our extraction procedure to extract the structure defined by *rep*. ABP' is this structure, restricted to β. We model-check ABP'; by Theorem 4, ABP is a refinement of ABP' and any $CTL^*\backslash X$ formulae that hold on ABP' also hold on ABP.

We also show that ABP' is WEB to a non-lossy protocol; in many cases such a check is more convincing than model-checking because it shows that one system is a refinement of another.

4.1 Well-Founded Equivalence Bisimulation

In this subsection we define a relation B and outline the ACL2 proof that B is a WEB. We start with some definitions.

Table 1. Rules defining the transition relation

Rule	Definition
Skip	$\underline{\textsf{skip}}$
Accept.m	$\neg svalid \ \rightarrow\ smsg, svalid \ :=\ m, \textsf{true}$
Send-msg	$svalid \ \rightarrow\ s2r \ :=\ s2r : \langle smsg, sflag\rangle$
Drop-msg	$s2r \neq \epsilon \ \rightarrow\ s2r \ :=\ tail.s2r$
Get-msg	$s2r \neq \epsilon \wedge \neg rvalid \ \rightarrow$
	$\underline{\textsf{if}}\ \ flag.head.s2r = rflag$
	$\quad \underline{\textsf{then}}\ \ s2r \ :=\ tail.s2r$
	$\quad \underline{\textsf{else}}\ \ s2r, rmsg, rvalid, rflag \ :=\ tail.s2r, info.head.s2r, \textsf{true}, flag.head.s2r$
Send-ack	$r2s \ :=\ r2s : rflag$
Drop-ack	$r2s \neq \epsilon \ \rightarrow\ r2s \ :=\ tail.r2s$
Get-ack	$r2s \neq \epsilon \ \rightarrow$
	$\underline{\textsf{if}}\ \ head.r2s = sflag$
	$\quad \underline{\textsf{then}}\ \ r2s, svalid, sflag \ :=\ tail.r2s, \textsf{false}, \neg sflag$
	$\quad \underline{\textsf{else}}\ \ r2s := \ tail.r2s$
Reply	$rvalid \ :=\ \textsf{false}$

For the following definitions, a and b are sequences of length 1, $a \neq b$, and x is an arbitrary finite sequence. The function *compress* acts on sequences to remove adjacent duplicates. Formally,

$$compress.\epsilon = \epsilon \qquad\qquad compress.a = a$$

$$compress.aax = compress.ax \qquad\qquad compress.abx = a : compress.bx$$

The predicate *good-s2r* recognizes sequences that define valid channel contents. Formally,

$$good\text{-}s2r.\epsilon = \textsf{true} \qquad\qquad good\text{-}s2r.ax = (a = \langle info.a, flag.a\rangle) \wedge good\text{-}s2r.x$$

The function *s2r-state* compresses the *s2r* field of a state, except that already received messages at the head of *s2r* are ignored. Formally,

$$s2r\text{-}state.s = compress.relevant\text{-}s2r.(s2r.s, \langle rmsg.s, rflag.s\rangle)$$

where the function *relevant-s2r* is defined by:

$$relevant\text{-}s2r.(\epsilon, a) = \epsilon \qquad\qquad relevant\text{-}s2r.(bx, a) = bx$$

$$relevant\text{-}s2r.(ax, a) = relevant\text{-}s2r.(x, a)$$

The function *r2s-state* compresses the *r2s* field of a state, except that acknowledgements at the head of *r2s* with a flag different from *sflag* are ignored. Formally,

$$r2s\text{-}state.s = compress.relevant\text{-}r2s.(r2s.s, sflag.s)$$

where the function *relevant-r2s* is defined by:

$$relevant\text{-}r2s.(\epsilon, a) = \epsilon \qquad\qquad relevant\text{-}r2s.(ax, a) = ax$$

$$relevant\text{-}r2s.(bx, a) = relevant\text{-}r2s.(x, a)$$

The main idea behind the bisimulation is to relate states that have similar compressed channels—*i.e.*, are equivalent under *s2r-state* and *r2s-state*—and are otherwise identical. We define the bisimulation in terms of rule

$$rep: \quad good\text{-}s2r.s2r \quad \rightarrow \quad s2r, r2s := s2r\text{-}state, r2s\text{-}state$$

We now define our proposed WEB B: sBu iff $rep.s = rep.u$. It is easy to see that B is an equivalence relation that, except for *s2r* and *r2s*, preserves the labeling of states. We define *rank*, a function on states as follows: $rank.s = |s2r.s| + |r2s.s|$.

We will show that $\langle rank, \langle \mathbb{N}, < \rangle\rangle$ is a well-founded witness (to be pedantic we can define *rank* so that it has two arguments, as follows: $rank.(u,s) = |s2r.s| + |r2s.s|$) Note that if sBw, sRu, and sBu, then uBw and by rule *Skip*, wRw, therefore, we need only concern ourselves with the case where $\neg sBu$. To show B is a WEB, it suffices to show:

$$sBw \wedge sRu \wedge \neg sBu \;\Rightarrow\; \langle \exists v : wRv : uBv \vee (sBv \wedge rank.v < rank.w)\rangle$$

We break up the proof (that B is a WEB) into the eight cases in Table 2 by expanding R, *i.e.*, by considering all the ways in which s can be related to u. The cases have the form: Rule Lemma; when u or v appear in Lemma they abbreviate the terms Rule.s and Rule.w, respectively. We prove the cases in ACL2.

Table 2. WEB case analysis

Rule	Lemma
Accept.m	$sBw \Rightarrow uBv$
Send-msg	$sBw \wedge \neg sBu \Rightarrow uBv$
Drop-msg	$sBw \wedge \neg sBu \Rightarrow (uBv) \vee (sBv \wedge rank.v < rank.w)$
Get-msg	$sBw \wedge \neg sBu \wedge u \neq \textit{Drop-msg}.s \Rightarrow (uBv) \vee (sBv \wedge rank.v < rank.w)$
Send-ack	$sBw \wedge \neg sBu \Rightarrow uBv$
Drop-ack	$sBw \wedge \neg sBu \Rightarrow (uBv) \vee (sBv \wedge rank.v < rank.w)$
Get-ack	$sBw \wedge \neg sBu \wedge u \neq \textit{Drop-ack}.s \Rightarrow (uBv) \vee (sBv \wedge rank.v < rank.w)$
Reply	$sBw \Rightarrow uBv$

In order to tie up the case analysis, we define a function *step* that takes three states, s, u, and w, as arguments. If sBu, *step* returns w, else if $u = A.s$, for A, a rule from Table 1, *step* returns $A.w$, else *step* returns w. Since we proved that B is an equivalence relation, the following theorem implies that B is a WEB (existential quantification is replaced by the witness function *step*):

$$sBw \wedge sRu \wedge v = step.(s,u,w) \quad\Rightarrow\quad wRv \wedge (uBv \vee (sBv \wedge rank.v < rank.w))$$

4.2 Quotient Extraction

In this subsection we prove the following ACL2 theorems which show that *rep* is a representative function satisfying the requirements of Theorem 3; hence, the quotient structure induced by *rep* is isomorphic to the quotient structure w.r.t. B: $sBw \equiv rep.s = rep.w$, $rep.rep.s = rep.s$, and $rank.rep.s \leq rank.s$. We extract the quotient structure (induced by *rep*) of the alternating bit protocol restricted to binary messages. In the following subsections, we describe the use of model-checking and WEB equivalence checking to analyze this structure.

We now have enough machinery to describe how refinement is used in the verification of the alternating bit protocol. *ABP* is the model of the alternating bit protocol in ACL2. ABP'' is *ABP* with *s2r*, *r2s* relabeled by *s2r-state* and *r2s-state*, respectively. B is a bisimulation on ABP'' with well-founded witness $\langle rank, \langle \mathbb{N}, < \rangle\rangle$, s.t. $rank.(u,s) = |s2r.f^{-1}.s| + |r2s.f^{-1}.s|$ (f is the bijection between *ABP* and ABP''; recall that *rank* is defined on states of ABP''). The quotient structure of ABP'' w.r.t. B is isomorphic to the structure induced by *rep*.

ABP' is this structure, with *s2r* and *r2s* hidden. It is ABP' that we analyze in the next two subsections. By Theorem 4, ABP is a refinement of ABP' and properties of ABP' can be lifted to ABP.

4.3 Model-Checking

We model-check the quotient structure extracted by the above mentioned procedure, using a μ-calculus model-checker and a fair-CTL to μ-calculus translator, both written in ACL2. We check the following formulae (written in $CTL^{*}\backslash X$):

1. $AG(sending1 \quad \Rightarrow \quad A(sending1 \;\; W \;\; rmsg = 1))$
2. $AG(receiving1 \quad \Rightarrow \quad A(receiving1 \;\; W \;\; delivered1))$
3. $AGEFsvalid$ (acceptance of a new message is always eventually possible)

where *sending1*, *receiving1*, and *delivered1*are abbreviations for $svalid \quad \wedge \quad smsg = 1$, $rvalid \ \wedge \ rmsg = 1$, and $\neg rvalid \ \wedge \ rmsg = 1$, respectively; formulae analogous to 1 and 2 are proved for message 0. All of the above formulae hold on the extracted structure, which is what one would expect. The property $AGAFsvalid$ (acceptance of a new message is always eventually guaranteed), however, does not hold without further fairness assumptions.

The liveness properties are as follows. Each property is shown under a set of fairness assumptions on the actions of the process. These are either weak fairness (infinitely often disabled or infinitely often executed) or strong fairness (infinitely often enabled implies infinitely often executed).

1. $AG(sendingNew1 \quad \Rightarrow \quad A(sending1 \;\; U \;\; rmsg = 1))$ (*sendingNew1* represents the sending of a new copy of message 1): This holds under weak fairness on the Send-msg and Reply actions, and strong fairness on the receipt of a new message by the action Get-msg. A similar property holds for message 0.
2. $AGAFsvalid$: This holds under the fairness assumptions for the previous property, along with weak fairness on the Send-ack action and strong fairness on the receipt of a new acknowledgment by the action Get-ack.

Since the fairness conditions mention actions, we compose Büchi automata accepting fair paths with the quotient structure and model-check the resulting structure on fair-CTL formulae which refer both to the propositions of the quotient structure and the accepting states of the automata.

We use an argument based on bisimulation to derive sufficient conditions for data-independence [Wol86] of the protocol. These are verified in ACL2; as a consequence, the properties shown above for the data domain $\{0, 1\}$ suffice to show similar properties for *arbitrary* data domains.

4.4 Bisimulation Checking

In many cases, the correctness proof is more convincing if we can show that the extracted model is bisimilar to a model that is so simple, it is correct by inspection. In the case of the alternating bit protocol, we can show that the extracted model is bisimilar to a simple, non-lossy version of the protocol, presented in Table 3.

We use a WEB equivalence checker (based on the description in [BCG88]) written in ACL2 to verify that the non-lossy protocol in Table 3 and the extracted protocol are WEB. The main idea is that we create the disjoint union of the transition systems corresponding to the extracted protocol and the non-lossy protocol. The algorithm will compute the coarsest WEB on a structure; hence, if the initial states of the two systems are in the same class, the two systems are WEB. In computing the coarsest WEB, we examine only *svalid*, *smsg*, *rvalid*, and *rmsg*. Notice that this view is exactly the one presented in Figure 1.

Table 3. Rules defining the transition relation of the non-lossy protocol

Rule	Definition
Accept.m	$\neg svalid \rightarrow smsg, svalid := m, \mathsf{true}$
Send-msg	$svalid \wedge \neg rvalid \wedge \neg sent \rightarrow rvalid, sent, rmsg := \mathsf{true}, \mathsf{true}, smsg$
Ready	$sent \rightarrow svalid, sent := \mathsf{false}, \mathsf{false}$
Reply	$rvalid := \mathsf{false}$

5 Related Work and Conclusions

Among related work, [MN95] prove safety properties of the alternating bit protocol by using Isabelle/HOL to prove that a manually constructed finite-state system contains all of the traces of the alternating bit protocol and then model-check the finite-state system. [HS96] show the correctness of an infinite-state system by using PVS to verify that a simple manually constructed finite-state system is a conservative approximation of the infinite-state system. The work described in this paper improves upon such methods by (i) using a (verified) representative function to *automatically* construct a quotient structure, and (ii) using WEBs instead of simulations or trace containment: this allows us to check properties *exactly*, *i.e.*, if a property holds (fails) on the simple system, then it holds (fails) on the original system.

There are several known types of infinite-state systems (*e.g.*, [ACD90,GS92,AJ96,EN95]) for which the model-checking problem is decidable, but these types of systems often turn out to be too specialized for many cases where it is possible to devise finite abstractions. There have been several approaches to automatically verifying the alternating bit protocol: safety properties of such lossy channel systems are decidable [AJ96]; however, in order to construct automatic abstractions that demonstrate liveness properties, most other verifications of the alternating bit protocol (*e.g.*, [GS97]) consider channels to be bounded.

Mechanical verification is necessary. In our case, we managed to convince ourselves that a candidate relation was a WEB for the alternating bit protocol, even though it was not; this became clear only when we tried to prove it mechanically.

An interesting direction for future work is to apply the methodology presented here to the verification of other infinite-state systems (*e.g.*, pipelined and out-of-order execution machines and memory coherence protocols).

Acknowledgments

J Moore was always available to discuss ACL2; he also read and commented on substantial parts of the proof script. Jun Sawada and Richard Trefler were involved in the early stages of the project; along with Rajeev Joshi, they have read this paper and have made many useful suggestions.

References

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model checking for real time systems. In *5th IEEE Symp. on Logic in Computer Science*, 1990.

[AJ96] P.A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2), 1996.

[BCG88] M. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59, 1988.

[BG94] M.A. Bezem and J.F. Groote. A correctness proof of a one bit sliding window protocol in mCRL. *The Computer Journal*, 1994.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDD's. In *Conference on Computer Aided Verification*, volume 1102 of *LNCS*, 1996.

[BM79] R. Boyer and J. Moore. *A Computational Logic.* Kluwer Academic Publishers, 1979.

[BSW69] K.A. Barlett, R.A. Scantlebury, and P.C. Wilkinson. A note on reliable full duplex transmission over half duplex links. In *Communications of the ACM*, volume 12, 1969.

[CE81] E.M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, volume 131 of *LNCS*. Springer-Verlag, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic. *ACM Transactions on Programming Languages and Systems*, 8(2), 1986.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *JACM*, 33(1):151–178, January 1986.

[EN95] E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *ACM Symposium on Principles of Programming Languages*, 1995.

[GM93] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[GS92] S. German and A.P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 1992.

[GS97] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Conference on Computer Aided Verification*, volume 1254 of *LNCS*, 1997.

[HS96] K. Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification. In *Formal Methods Europe (FME)*, volume 1051 of *LNCS*. Springer-Verlag, 1996.

[KM97] M. Kaufmann and J S. Moore. An industrial strength theorem prover for a logic based on Common Lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213, April 1997.

[Lam80] L. Lamport. "Sometimes" is sometimes "not never". In *ACM Symposium on Principles of Programming Languages*, 1980.

[Mil90] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1990.

[MN95] O. Müller and T. Nipkow. Combining model checking and deduction for I/O-Automata. In *Proceedings of TACAS*, 1995.

[Nam97] K. S. Namjoshi. A simple characterization of stuttering bisimulation. In *17th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1346 of *LNCS*, pages 284–296, 1997.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. of the 5th International Symposium on Programming*, volume 137 of *LNCS*, 1982.

[Wol86] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proceedings of the 13th ACM Symposium on Principles of Programming Languages*, pages 184–193. ACM Press, 1986.