... Symposium on Logic in Computer Science, June 1987,
Ithaca, New York, IEEE Computer Society.

Proving Boolean Combinations of Deterministic Properties

Bowen Alpern
Watson Research Center
Box 218
Yorktown Heights, NY 10598

Fred B. Schneider*
Department of Computer Science
Cornell University
Ithaca, NY 14853

Abstract This paper gives a method for proving that a program satisfies a temporal property that has been specified in terms of Buchi automata. The method permits extraction of proof obligations for a property formulated as the Boolean combination of properties, each of which is specified by a deterministic Buchi automaton, directly from the individual automata. The proof obligations can be formulated as Hoare triples. The method is proved sound and relatively complete. A simple example illustrates application of the method.

1. Introduction

Over the past 20 years, there has been increasing interest in ways to deduce properties of program behavior from the program text itself. One technique that is attracting attention is based on specifying a temporal property P with a Buchi automata that accepts every sequence of program states satisfying P.[1] Proof obligations are then extracted from this automaton. These obligations are generalizations of the invariant and variant function used to prove partial correctness and termination of a while loop. They define verification conditions that must hold for any program satisfying P. Since the verification conditions themselves are formulated as Hoare triples [Hoare 69], all reasoning can be done in first-order predicate logic, and temporal logic is not required.

Extraction of proof obligations from Buchi automata was first proposed in [Alpern & Schneider 85] for *deterministic properties*—those properties that can be specified by deterministic Buchi automata. Subsequently, [Manna & Pnueli 87] generalized the approach to handle properties that can be specified by non-deterministic Buchi automata.[2] Both [Alpern & Schneider 85] and [Manna & Pnueli 87], however, treat a property as a monolithic entity. Doing so has some practical disadvantages. First, the Buchi automaton specifying a property that is the Boolean combination of simpler properties can be exponentially bigger than the combined size of the automata specifying its components [Sistla et al 85]. Since the amount of work required to prove correctness is linear in the size of the automata specifying the property being proved, working with the combined automata can be impractical. Second, combining properties requires complex automata constructions and simplifying properties requires automata transformations, both of which are error prone processes. By contrast, when, for example, temporal logic is used to specify properties, logical connectives permit properties to be combined and the laws of logic allow properties to be simplified.

This paper shows how to avoid these disadvantages by giving a way that proof obligations for a property formulated as the Boolean combination of deterministic properties can be extracted directly from the automata specifying those components. The method also unifies the approaches in [Alpern & Schneider 85] and [Manna & Pnueli 87]. The method of [Alpern & Schneider 85] extracts obligations for P from a deterministic Buchi automata that accepts P; the method of [Manna & Pnueli 87] is equivalent to extracting obligations for $\neg P$ from a Buchi automata that accepts P.

In our new approach, a property is presented as a conjunction of *clauses*, each of which is proved separately and is formulated as the disjunction of deterministic properties and negations of deterministic properties. To prove that a program satisfies a clause, three *proof instruments* must be exhibited. An invariant handles the safety aspects of the proof; a variant function handles the liveness aspects; and a *candidate function*, arbitrates among the automata specifying the disjuncts of a clause to ensure that any program execution will be accepted by at least one automata representing a deterministic property or rejected by at least one automata representing the negation of a deterministic property. The three instruments define verification conditions, which can be formulated as Hoare triples.

In section 2, a simple concurrent programming language and its semantics is defined. Section 3 reviews Buchi automata for specifying properties. Proof instruments for a single clause are discussed in section 4; sound-

*Supported, in part, by NSF Grant DCR-8320274 and Office of Naval Research contract N00014-86-K-0092.

[1]Buchi automata are expressive enough for use in proving all linear-time temporal properties described in [Manna & Pnueli 81], [Owicki & Lamport 82], etc.

[2]Non-deterministic Buchi automata can specify properties that cannot be specified using deterministic Buchi automata [Eilenberg 74].

ness and completeness proofs are also given there. The method is illustrated in section 5 by proving that assuming weak fairness, Peterson's protocol for mutual exclusion [Peterson 81] satisfies a non-starvation property. Section 6 relates our approach to other techniques for proving properties of concurrent programs.

2. Programs

A program π is specified by a predicate $Init_\pi$ describing possible initial states and a set of atomic actions A_π. An execution of π is modeled by an infinite sequence of program states, called a *history*, in which the first state satisfies $Init_\pi$ and every subsequent state results from executing an element of A_π in the preceding state. In a concurrent or distributed program, a history is the sequence of states that results from interleaving the atomic actions of processes as they are executed.

An atomic action α defines a set of pairs of program states. Execution of α is *enabled* in a state s, denoted $enabled(\alpha, s)$, provided $(\exists\ t\colon\ \langle s, t\rangle \in \alpha)$. The statement

$$\langle \textbf{if}\ b \rightarrow S\ \textbf{fi}\rangle$$

specifies an atomic action containing those elements $\langle s, t\rangle$ such that b holds on s and t is the state produced by executing S starting in state s.

Rather than enumerating the atomic actions of a program directly, it is frequently convenient to annotate a program with control points that delimit its atomic actions. We will denote control points by marking and numbering them. For example, program fragment of process π

$$\ldots\ \{3{:}\}\ S\ \{4{:}\}\ \ldots$$

defines a single atomic action

$$\langle \textbf{if}\ pc_\pi = 3 \rightarrow pc_\pi := 4;\ S\ \textbf{fi}\rangle$$

where pc_π simulates the program counter.

All histories are assumed to be infinite. One way to achieve this is by including in A_π an atomic action that has no effect on the program state and is enabled when no other atomic action is. Thus, terminating executions are extended to infinite histories by repeating the final state.

An example program *MEP* is shown in Figure 2.1. It is a solution to the two-process critical section problem described in [Peterson 81]. Based on the control point annotations, we obtain set of atomic actions A_{MEP} of Figure 2.2. In those atomic actions, variable pc_A simulates the program counter for process A and variable pc_B simulates the program counter for process B. Finally, we have

$$Init_{MEP}\colon\ pc_A = 1\ \wedge\ pc_B = 1$$

because when execution is begun, both processes start at the the beginning of their loops.

3. Specifying Properties using Buchi Automata

A Buchi automaton is a finite state machine that accepts infinite sequences. Since a property of a program can be viewed as defining a set of infinite sequences of program states, a Buchi automaton m can be used to specify the property containing exactly those sequences accepted by m. Mechanical procedures exist to translate linear-time first-order temporal formulas into Buchi automata where state transitions are defined in terms of first-order program state predicates [Wolper et al 83] [Alpern 86]. Moreover, Buchi automata have natural diagrammatic representations and this is sometimes convenient.

```
MEP: cobegin
  A :: do true → {1: } ncs_A;
                 {2: } active_A := true;
                 {3: } turn := B;
                 {4: } do active_B ∧ turn=B → {5: } skip
                       od;
                 {6: } cs_A;
                 {7: } active_A := false
       od
  //
  B :: do true → {1: } ncs_B;
                 {2: } active_B := true;
                 {3: } turn := A;
                 {4: } do active_A ∧ turn=A → {5: } skip
                       od;
                 {6: } cs_B;
                 {7: } active_B := false
       od
  coend
```

Figure 2.1. Peterson's Mutual Exclusion Protocol

$$
\begin{array}{rl}
A_{MEP} = \{ & \alpha_1^A\colon\ \langle \textbf{if}\ pc_A = 1 \rightarrow pc_A := 2;\ ncs_A\ \textbf{fi}\rangle,\\
& \alpha_2^A\colon\ \langle \textbf{if}\ pc_A = 2 \rightarrow pc_A,\ active_A := 3,\ true\ \textbf{fi}\rangle,\\
& \alpha_3^A\colon\ \langle \textbf{if}\ pc_A = 3 \rightarrow pc_A,\ turn := 4,\ B\ \textbf{fi}\rangle,\\
& \alpha_4^A\colon\ \langle \textbf{if}\ pc_A = 4 \wedge active_B \wedge turn = B\\
& \qquad \rightarrow pc_A := 5\ \textbf{fi}\rangle,\\
& \alpha_5^A\colon\ \langle \textbf{if}\ pc_A = 4 \wedge \neg(active_B \wedge turn = B)\\
& \qquad \rightarrow pc_A := 6\ \textbf{fi}\rangle,\\
& \alpha_6^A\colon\ \langle \textbf{if}\ pc_A = 5 \rightarrow pc_A := 4\ \textbf{fi}\rangle,\\
& \alpha_7^A\colon\ \langle \textbf{if}\ pc_A = 6 \rightarrow pc_A := 7;\ cs_A\ \textbf{fi}\rangle,\\
& \alpha_8^A\colon\ \langle \textbf{if}\ pc_A = 7 \rightarrow pc_A,\ active_A := 1,\ false\ \textbf{fi}\rangle,\\
& \alpha_1^B\colon\ \langle \textbf{if}\ pc_B = 1 \rightarrow pc_B := 2;\ ncs_B\ \textbf{fi}\rangle,\\
& \alpha_2^B\colon\ \langle \textbf{if}\ pc_B = 2 \rightarrow pc_B,\ active_B := 3,\ true\ \textbf{fi}\rangle,\\
& \alpha_3^B\colon\ \langle \textbf{if}\ pc_B = 3 \rightarrow pc_B,\ turn := 4,\ A\ \textbf{fi}\rangle,\\
& \alpha_4^B\colon\ \langle \textbf{if}\ pc_B = 4 \wedge active_A \wedge turn = A\\
& \qquad \rightarrow pc_B := 5\ \textbf{fi}\rangle,\\
& \alpha_5^B\colon\ \langle \textbf{if}\ pc_B = 4 \wedge \neg(active_A \wedge turn = A)\\
& \qquad \rightarrow pc_B := 6\ \textbf{fi}\rangle,\\
& \alpha_6^B\colon\ \langle \textbf{if}\ pc_B = 5 \rightarrow pc_B := 4\ \textbf{fi}\rangle,\\
& \alpha_7^B\colon\ \langle \textbf{if}\ pc_B = 6 \rightarrow pc_B := 7;\ cs_B\ \textbf{fi}\rangle,\\
& \alpha_8^B\colon\ \langle \textbf{if}\ pc_B = 7 \rightarrow pc_B,\ active_B := 1,\ false\ \textbf{fi}\rangle\ \}
\end{array}
$$

Figure 2.1. Atomic Actions A_{MEP}

Figure 3.1 is an example of a Buchi automaton m_{ae}. It accepts an infinite sequence in which after a finite prefix each state satisfies the program state predicate P. In temporal logic, this property is given by $\Diamond\Box P$. Automaton m_{ae} contains three *automaton states* labeled q_0, q_1, and q_2. The *start state*, q_0, is denoted by an edge with no origin, and *accepting state*, q_1, is denoted by two concentric circles. A Buchi automaton accepts a sequence σ if and only if it enters an accepting state infinitely often while reading σ. Notice, there is no way in m_{ae} to get from q_2 to an accepting state. Such states are called *dead states*. If an automaton is in a dead state, it cannot accept its input.

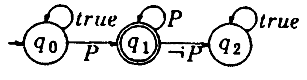

Figure 3.1. m_{ae}

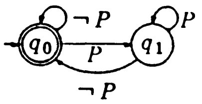

Figure 3.2. m_{io}

Edges between automaton states are labeled by program-state predicates that are called *transition predicates* and define transitions between automaton states. If a program state satisfies the transition predicate on an edge then that edge is defined for that program state. For example, because there is an edge labeled P from q_0 to q_1 in m_{ae}, whenever m_{ae} is in q_0 and the next symbol read is a program state satisfying P, then a transition to q_1 can be made. We adopt the convention that there be an edge defined from each automaton state for each input symbol.

In order to define Buchi automata formally, the following notation will be useful. For any sequence $\sigma = s_0 s_1 \ldots$,

$\sigma[i]$ = s_i,
$\sigma[..i]$ = $s_0 s_1 \ldots s_{i-1}$,
$\sigma[i..]$ = $s_i s_{i+1} \ldots$,
$|\sigma|$ = the length of σ (ω if σ is infinite),
$INF(\sigma)$ = $\{s | s$ appears infinitely often in $\sigma\}$.

A Buchi automaton m for a property of a program π is a five-tuple $\langle S_\pi, Q, Q_0, A, \delta\rangle$, where

S_π is the (countable) set of program states of π,
Q is the (finite) set of automaton states of m,
$Q_0 \subseteq Q$ is the set of start states of m,
$A \subseteq Q$ is the set of accepting states of m,
$\delta \in (Q \times S) \rightarrow 2^Q - \varnothing$ is the *transition function* of m.

Transition predicate T_{ij} associated with the edge from automaton state q_i to q_j, holds for a program state s if and only if $q_j \in \delta(q_i, s)$. (Although S_π may be infinite, we only consider properties for which the transition predicates have finite description as a first-order formula involving program variables.) Transition function δ can be extended to handle finite sequences of program states in the usual way:

$$\delta^*(q, \sigma) = \begin{cases} \{q\} & \text{if } |\sigma| = 0 \\ \{q' | \; \exists q'': q'' \in \delta^*(q, \sigma[..|\sigma|-1]): \\ \quad q' \in \delta(q'', \sigma[|\sigma|-1])\} & \text{if } 0 < |\sigma| < \omega \end{cases}$$

A sequence of automaton states that m might occupy while reading σ is called a *run*. Thus, ρ is a run of m on σ if and only if

$$\rho[0] \in Q_0 \text{ and } (\forall i: 0 < i < |\sigma|: \rho[i] \in \delta(\rho[i-1], \sigma[i-1])).$$

A Buchi automaton m accepts a sequence σ if and only if there is a run ρ of m on σ for which $INF(\rho) \cap A \neq \varnothing$.

Notice in m_{ae} (Figure 3.1) that two transitions are possible from q_0 for a program state satisfying P, because any program state that satisfies P also satisfies *true*. When there is more than one start state or more than one transition is possible from some automaton state for a given input symbol, the automaton is *non-deterministic*; otherwise, it is *deterministic*. Thus, m_{ae} is non-deterministic. However, $L(m_{ae})$ the language accepted by m_{ae} is the negation of the deterministic property $\Box\Diamond\neg P$ accepted by m_{io} of Figure 3.2.

4. Proof Instruments from Buchi Automata

Let property P be the conjunction of clauses $P_1, P_2, \ldots P_m$. To prove that a program π satisfies P, we prove separately that π satisfies each of the clauses. Thus, it suffices to derive proof instruments for a single clause P_i of the form

$$D_1 \vee \ldots \vee D_p \vee \neg D_{p+1} \vee \ldots \vee \neg D_{p+n},$$

where each D_k is specified by m_k, a deterministic Buchi automaton. Any property that can be specified by a non-deterministic Buchi automaton can be put in this form [Eilenberg 74]. We call m_1 through m_p the *positive automata* of P_i and m_{p+1} through m_{p+n} the *negative automata*. Let m_k be defined by $\langle S_\pi, Q_k, \{q_{k0}\}, A_k, \delta_k\rangle$.

To formulate proof instruments for π and P_i, define a *joint state* x to be an element of $JS(P_i, \pi)$ the *joint state space*, $Q_1 \times \cdots \times Q_{p+n} \times S_\pi$. Let $x^{(k)}$ denote the k^{th} element of a joint state x and let $x^{(\pi)}$ abbreviate $x^{(p+n+1)}$. A joint state x is *positive*, denoted $Pos(x)$, iff for some positive automata m_j, $x^{(j)} \in A_j$. Thus, x is positive if one of m_1 through m_p is in an accepting state. The set of negative automata that are in an accepting state in a joint state x is $neg(x) = \{k \,|\, p < k \leq p+n \;\wedge\; x^{(k)} \in A_k\}$. For ρ a sequence of joint states, define $\rho^{(k)}$ to be the projection of ρ onto its k^{th} element; i.e. $\rho^{(k)} = \rho[0]^{(k)}\rho[1]^{(k)}\ldots$. A sequence of joint states ρ is a *joint history* of π and P iff $\rho^{(\pi)}$ is a history of π and for all $1 \leq k \leq p+n$, $\rho^{(k)}$ is a run of m_k on $\rho^{(\pi)}$.

A program π satisfies the property specified by a clause P_i iff each history σ of π is either accepted by one of the positive automata—and therefore satisfies one of the properties D_1 through D_p—or is rejected by one of the negative automata—and therefore satisfies one of $\neg D_{p+1}$ through $\neg D_{p+n}$. Thus, π satisfies P_i iff for every history σ there is a joint history ρ such that $\sigma = \rho^{(\pi)}$ and either (i) there is a positive automata m_j, such that $INF(\rho^{(j)}) \cap A_j \neq \varnothing$ or (ii) there is a negative automata m_k such that $INF(\rho^{(k)}) \cap A_k = \varnothing$.

To prove that π satisfies the property specified by a clause P_i, three proof instruments must be exhibited:

- an *invariant* $I \subseteq JS(P_i, \pi)$,
- a *candidate function* $u: JS(P_i, \pi) \rightarrow 2^{[p+1..p+n]}$, and
- a *variant function* $v: JS(P_i, \pi) \rightarrow W$, where W is a well-founded set.

The invariant relates program states in a history to automaton states occupied while reading that history. The candidate function identifies negative automata that might never again enter an accepting state. And, the variant function bounds the number of times that the candidate function can become empty before one of the positive automata enters an accepting state.

If the invariant, candidate function, and variant function satisfy the following obligations, then π will satisfy P_i. In these obligations, x and x' denote elements of $JS(P_i, \pi)$.

The first two obligations ensure that I holds throughout any joint history.

O1: If $x = \rho[0]$ for some joint history of π and P_i then $x \in I$.

O2: If $x \in I$ and $\langle x^{(\pi)}, x'^{(\pi)} \rangle \in \alpha$ then $x' \in I$.

O1 constitutes the base case and O2 an induction step for an argument that execution of each atomic action produces a joint state in I. Choosing an invariant I permits unreachable joint states to be ignored in the obligations that follow.

The next obligation requires that a variant function v increase only in or immediately after processing a joint state in which a positive automata is in an accepting state.

O3: If $x \in I$ and $\langle x^{(\pi)}, x'^{(\pi)} \rangle \in \alpha$ then $Pos(x) \vee Pos(x') \vee v(x) \leq v(x')$.

Thus, O3 ensures that either (i) some positive automaton enters an accepting state infinitely often (and the history is accepted) or (ii) after some point v never increases and no positive automaton subsequently enters an accepting state.

Since the range of v is a well-founded set, if v never increases then eventually its value must become constant. The next obligation implies that under these circumstances, throughout the rest of the history, the candidate function is a non-empty set and contains the index of some negative automaton m_k.

O4: If $x \in I$ and $\langle x^{(\pi)}, x'^{(\pi)} \rangle \in \alpha$ then $Pos(x) \vee Pos(x') \vee v(x) < v(x') \vee \emptyset \subset u(x') \subseteq u(x)$.

The final obligation ensures that a negative automata that enters an accepting state cannot be a candidate unless a positive state is entered or the variant function decreases. Thus, if no positive state is entered and the variant function does not decrease then m_k does not enter an accepting state for the remainder of the history, hence it rejects the history.

O5: If $x \in I$ and $\langle x^{(\pi)}, x'^{(\pi)} \rangle \in \alpha$ then $Pos(x) \vee Pos(x') \vee neg(x') \cap u(x') = \emptyset$.

Since m_k rejects the history, the history satisfies $\neg D_k$ and therefore satisfies P_i.

We can now prove:

Soundness Theorem: If there are proof instruments I, v, and u that satisfy obligations O1 through O5 then π satisfies P_i.

Proof: Let ρ be a joint history of π and P_i. $\rho[0] \in I$ by O1. By O2 and induction, $\rho[j] \in I$ for all $j \geq 0$. We must show that ρ causes some positive automaton to be in an accepting state infinitely often or some negative automaton to be in an accepting state only finitely often. Assume that no positive automaton is in an accepting state infinitely often. Thus, there is an index l_1 such that $\rho[l_1..]$ contains no positive states. By O3, the variant function is non-increasing on $\rho[l_1..]$. Since its range is well-founded, there must be an index l_2 such that $l_1 \leq l_2$ and the variant function is constant on $\rho[l_2..]$. By O4, there is a negated automaton m_k such that $k \in u(\rho[l_2])$. Also by O4, for all $l_2 < j$, $k \in u(\rho[j])$. Therefore, by O5, m_k does not enter an accepting state after l_2. This means that m_k rejects ρ, so π satisfies P_i.

We now show that our technique is relatively complete.

Completeness Theorem: If π satisfies P then there exist proof instruments I, v, and u that satisfy obligations O1 through O5.

Proof: Form a directed graph where the nodes are the joint states appearing in joint histories and there is an edge from node x to y iff y is not a positive accepting state and x immediately precedes y in some joint history. Define partial order $\gg$ on the nodes of this graph such that $x \gg y$ iff there is a path with non-zero length from x to y and somewhere on this path appears an accepting state for each negative automaton.

Partial order $\gg$ is well-founded, as is shown by the following proof by contradiction. If $\gg$ were not well-founded then there would be an infinite descending chain $x_1 \gg x_2 \ldots$. By construction of the graph, this implies the existence of a joint history that includes x_1. Let σ_0 be a prefix of such a joint history that ends with x_1. For each x_i in the infinite descending chain, let σ_i be the path from x_i to x_{i+1} that includes an accepting state for every negative automaton. Such a path exists by definition, because $x_i \gg x_{i+1}$. Finally, let σ be $\sigma_0\sigma_1\ldots$. Notice that σ is a joint history, that σ contains no positive accepting states after σ_0, and that there are infinitely many accepting states for each negative automaton in σ. Thus, σ does not satisfy P. This is a contradiction, and we conclude that $\gg$ is well-founded.

Since $\gg$ is well-founded, the following recursive function is well defined:

$$H(x) = \sup_{x \gg y}\{H(y)+1\}.$$

H is a function from the joint state space to the ordinals. If there is no y such that $x \gg y$ then $H(x)=0$ by definition of *sup*, so the function is total. Notice that if $x \gg y$ then $H(x) > H(y)$. Moreover, if there is any path from x to y in the graph then $H(x) \geq H(y)$.

The variant function will be constructed using H and the *level* $l(x)$ of a node x, defined as follows. Level $l(x) = i$ iff for any collection of i and no more than i negative automata, there exists some node w with $H(w)=H(x)$ such that there is a path to x to w and there is an accepting state for each automaton in the collection somewhere on the path. Note that by definition of H, the level of a node will be less than n.

The three proof instruments can now be defined. Choose I to be the characteristic predicate for the set of joint states that appear in joint histories of π and P_i. Choose $v(x)$ to be $\langle H(x), n-l(x) \rangle$. The range of v with lexicographic ordering of pairs as the ordering relation is well-founded because the ordinals are. Finally, choose $u(x)$ to be the set of negative automata that do not have accepting states on any path from w to x for any $w \neq x$ such that $v(w)=v(x)$.

Proof obligations O1 and O2 follow immediately from the definition of I.

To see that O3 holds, notice that if neither x nor x' is positive then there is a path (of length 1) from x to x' in the graph. Thus, $H(x) \geq H(x')$. Suppose $H(x)=H(x')$. Since any path to x can be extended to x', $l(x) \leq l(x')$. Therefore, $v(x) \geq v(x')$.

To see that O4 holds, suppose that neither x nor x' is positive and that $v(x)=v(x')$. By construction of v, both x and x' must be at the same level. By the construction of u, $u(x') \subset u(x)$, since there is a path (of length 1) from x to x'. Also by construction of u, $u(x')\neq\varnothing$.

If there exists a node w such that $v(w)=v(x')$ and there is a path from w to x' then $neg(x')\cap u(x')=\varnothing$ since x' is (trivially) included on the path to x'. If there is no such w then $v(x)>v(x')$. So, O5 holds. This completes the proof.

Obligations O1 through O5 can be translated into verification conditions formulated as Hoare triples and first-order predicate logic formulas. A Hoare triple $\{P\}\ \alpha\ \{Q\}$ for an atomic action α asserts that any execution of α started in a state satisfying P terminates in a state satisfying Q.[3] Thus, $\{P\}\ \alpha\ \{Q\}$ is valid iff $(\forall\langle s,t\rangle\colon\ \langle s,t\rangle\in\alpha\colon\ P(s)\Rightarrow Q(t))$.

To reformulate O1 through O5 in terms of Hoare triples, we define slight variations of the three proof instruments. Define $PJS(P_i)$ the projection of $JS(P_i,\pi)$ with respect to program states as $PJS(P_i)=Q_1\times\cdots\times Q_{p+n}$. Elements of $PJS(P_i)$ are called *projected joint states*. The projected joint state in which every automaton is in its start state is called the *projected joint start state*. A projected joint state y is considered *positive* if it is the projection of a positive state. (Similarly, if y is a projection of joint state x then define $neg(y)=neg(x)$.) We now define projections of the three proof instruments with respect to projected joint states. For an invariant I, I_y is the projection of I with respect to a projected joint state y; i.e., $I_y=\{s\mid s\in S_\pi\wedge y\times s\in I\}$. For a variant function v, projection v_y is a function from S_π to W such that $v(y,s)=v_y(s)$ for $s\in S_\pi$. And, for a candidate function u, define u_y such that $u(y,s)=u_y(s)$ for $s\in S_\pi$.

Finally, we define a *projected joint state transition predicate* T_{yz} to be a predicate that holds for any program state causing a transition from a projected joint state y to a projected joint state z. T_{yz} can be formed by taking the conjunction for all m_i of the transition predicates labeling the edge from $y^{(i)}$ to $z^{(i)}$. T_{0z} is the projected joint state transition predicate from the joint start state to z.

Verifying each of the following formulas implies that O1 through O5 hold. The first verification condition implies O1.

VC1: $(Init_\pi\ \wedge\ T_{0z})\ \Rightarrow\ I_z$

The next implies O2 through O5.

VC2: For all projected joint states y and z and atomic actions $\alpha\in A_\pi$:

(i) If y or z is positive then $\{I_y\}\ \alpha\ \{T_{yz}\Rightarrow I_z\}$.

(ii) Otherwise,

$$\begin{array}{l}\{I_y\ \wedge\ v_y=\mathrm{V}\ \wedge\ u_y=\mathrm{U}\}\\ \alpha\\ \{T_{yz}\ \Rightarrow\ (I_z\ \wedge\ (v_z<\mathrm{V}\\ \quad\vee\ (v_z=\mathrm{V}\wedge\varnothing\subset u_z\subseteq\mathrm{U}-neg(z))))\}\end{array}$$

O2 is implied by $T_{yz}\Rightarrow I_z$ in the postcondition of VC2(i) and VC2(ii). To see that O3, O4, and O5 follow from VC1 and VC2, choose

$x=\langle y,s\rangle$ and $x'=\langle z,s'\rangle$

and assume $x\in I$ and $\langle s,s'\rangle\in\alpha$. It follows immediately from O2 that $x'\in I$. We conclude that s' satisfies T_{yz} directly from the definition of T_{yz}. If y or z is positive, then O3, O4, and O5 are trivially satisfied. Suppose neither y nor z is positive and let $v(x)=\mathrm{V}$ and $u(x)=\mathrm{U}$. From the postcondition of VC2(ii), it follows that either $v(x')<\mathrm{V}$ or $v(x')=\mathrm{V}$. Thus, O3 holds. If $v(x')<\mathrm{V}$ does not hold then, from the postcondition of VC2(ii), $\varnothing\subset u(x')\subseteq\mathrm{U}$ and $u(x')\subseteq\mathrm{U}-neg(x')$. Thus, O4 and O5 hold.

Conversely, assume O1 through O5 hold. VC1 follows from O1. From O2 we get

$\{I_y\}\ \alpha\ \{T_{yz}\Rightarrow I_z\}$.

Thus, VC2(i) holds. Assume that neither y nor z is positive. From O3 we get

$\{I_y\ \wedge\ v_y=\mathrm{V}\}\ \alpha\ \{T_{yz}\Rightarrow v_z\leq\mathrm{V}\}$.

From O4 we get

$\{I_y\ \wedge\ v_y=\mathrm{V}\ \wedge\ u_y=\mathrm{U}\}\ \alpha\ \{T_{yz}\Rightarrow(v_z<\mathrm{V}\vee\varnothing\subset u_z\subseteq\mathrm{U})\}$.

Finally, from O5 we get

$\{I_y\ \wedge\ v_y=\mathrm{V}\ \wedge\ u_y=\mathrm{U}\}\ \alpha\ \{T_{yz}\Rightarrow(v_z<\mathrm{V}\vee u_z\subseteq\mathrm{U}-neg(z))\}$.

Together, these four Hoare triples imply VC2(ii).

We have shown that obligations O1 through O5 are equivalent to verification conditions VC1 and VC2. If the underlying first-order predicate logic is expressive enough to capture the preconditions and postconditions of the Hoare triples of VC2 then the verification conditions can be expressed in this logic. Since Hoare Logic is semantically complete relative to the completeness of the underlying first-order predicate logic [Apt 81], our proof technique is complete relative to the semantic and expressive completeness of this logic.

5. Example: Proving Non-Starvation Assuming Fairness

In addition to mutual exclusion, a solution to the critical section problem should ensure that processes attempting entry to critical sections actually do enter eventually. A process of *MEP* (Figure 2.1) is said to *starve* if it trys to enter its critical section but never succeeds. To formalize this, let pc_A (pc_B) be a variable containing the value of the program counter for process A (B). Process A of *MEP* should eventually enter cs_A whenever

$TRY_A\colon\ pc_A=4\ \vee\ pc_A=5$

holds, and similarly process B should eventually enter cs_B whenever

$TRY_B\colon\ pc_B=4\ \vee\ pc_B=5$

holds. The Non-Starvation Property NS_{MEP} asserts that neither process starves. This can be formalized in temporal logic as

$NS_{MEP}\colon\ (\Box\Diamond\neg TRY_A)\ \wedge\ (\Box\Diamond\neg TRY_B)$.

It is easy to see that *MEP* does not satisfy NS_{MEP} for histories in which one or the other process is not given sufficient opportunity to execute. A *fairness assumption* asserts that an atomic action that is enabled "often enough" will be executed eventually. Let F_α be a fairness assumption for atomic action α.[4] A fairness assumption for a program π is the conjunction of fairness assumptions—one for each atomic action in A_π. Thus, F_{MEP} the fairness assumption for *MEP* is defined by

[3]Since α is an atomic action, it *must* terminate once execution commences.

[4]See [Francez 86] for various types of fairness assumptions.

F_{MEP}: $\bigwedge_{\alpha \in A_{MEP}} F_\alpha$.

To show that MEP satisfies NS_{MEP} for all histories satisfying F_{MEP}, we must prove that MEP satisfies $F_{MEP} \Rightarrow NS_{MEP}$. Putting this property into conjunctive normal form, we get

$$FNS_{MEP}: \ (\bigvee_{\alpha \in A_{MEP}} \neg F_\alpha \ \vee \ \Box\Diamond\neg TRY_A) \ \wedge \ (\bigvee_{\alpha \in A_{MEP}} \neg F_\alpha \ \vee \ \Box\Diamond\neg TRY_B)$$

Each of the two clauses (conjuncts) in FNS_{MEP} is proved separately. However, since the two clauses are symmetric, only the first one is proved here; the proof of the second is similar.

As a fairness assumption for atomic actions, we choose *weak fairness*, which asserts that an atomic action that becomes enabled is eventually executed. This is expressed in temporal logic as

$$WF_\alpha: \ \Box\Diamond\neg Before_\alpha.$$

where $Before_\alpha(s)$ is the predicate $(\exists\ t\colon \langle s,\ t\rangle \in \alpha)$. Thus, the first conjunct of FNS_{MEP} property to be proven is

$$FNS^A_{MEP}: \ \bigvee_{\alpha \in A_{MEP}} \neg WF_\alpha \ \vee \ \Box\Diamond\neg TRY_A$$

$$= \ \bigvee_{\alpha \in A_{MEP}} \neg\Box\Diamond\neg Before_\alpha \ \vee \ \Box\Diamond\neg TRY_A$$

This clause contains seventeen disjuncts because $|A_{MEP}| = 16$; all but one of these disjuncts is negated. Each disjunct is of the form $\Box\Diamond\neg P$, so the Buchi automaton specification for each literal is m_{i_o} of Figure 3.2.

To prove that FNS^A_{MEP} holds we use the following proof instruments.

$$I \ = \ JS(FNS^A_{MEP},\ MEP)$$

$$v \ = \ \begin{cases} 0 & \neg TRY_A \\ pc_A - 3 & TRY_A \wedge TRY_B \wedge turn = A \\ 2 + (4 - pc_B) \bmod 7 & TRY_A \wedge \neg TRY_B \\ 7 + pc_B - 3 & TRY_A \wedge TRY_B \wedge turn = B \end{cases}$$

$$u \ = \ \begin{cases} \varnothing & \neg TRY_A \\ \{i \mid enbl(\alpha_i^A)\} & TRY_A \wedge TRY_B \wedge turn = A \\ \{i \mid enbl(\alpha_i^B)\} & TRY_A \wedge \neg TRY_B \\ \{i \mid enbl(\alpha_i^B)\} & TRY_A \wedge TRY_B \wedge turn = B \end{cases}$$

where $enbl(\alpha)$ holds in any program state s where atomic action α is enalbed (i.e., $enabled(\alpha,\ s)$).

Rather than checking that verification conditions VC1 and VC2 hold, we argue (informally) that obligations O1–O5 are satisfied by this choice of proof instruments.

Clearly, O1 and O2 are satisfied by the choice of invariant, since the invariant rules out no joint state.

O3 follows from the construction of the variant function, as follows. When $\neg TRY_A$, $v = 0$, but an automaton for a positive disjunct is in an accepting state, so O3 is satisfied in that case. If $TRY_A \wedge TRY_B \wedge turn = A$, then v is the number of atomic actions that A must execute before entering cs_A. Since $turn = A$, subsequent execution by B does not alter this value. If $TRY_A \wedge \neg TRY_B$ then v includes the number of atomic actions B must execute to establish the previous case $(TRY_A \wedge TRY_B \wedge turn = A)$. Clearly, execution by A does not alter this value. Finally, when $TRY_A \wedge TRY_B \wedge turn = B$, then v is the maximum value of v in the preceding case $(TRY_A \wedge \neg TRY_B)$ plus the number of atomic actions that B must execute to make $TRY_A \wedge \neg TRY_B$ hold.

O4 follows from the construction of the candidate function, which always contains the index of the automaton corresponding the fairness assumption for the atomic action that will reduce v.

O5 holds because m_{i_o} for an enabled atomic action is not in an accepting state.

Note that this proof is an instance of the method of helpful directions for weak fairness [Francez 86]. Each process corresponds to a direction. Execution of an enabled atomic action in the helpful direction decreases the variant function. Execution of an enabled atomic action in some other direction does not increase the variant function nor does it disable a helpful atomic action. Thus, when $TRY_A \wedge TRY_B \wedge turn = A$, the helpful direction is process A; when $TRY_A \wedge \neg TRY_B$ or $TRY_A \wedge TRY_B \wedge turn = B$ the helpful direction is process B. The weak fairness assumption guarantees that the variant function decreases eventually.

The method of helpful directions is a special case (for proving properties that assume weak fairness) of the technique developed in this paper. To see this observe that the candidate function identifies the helpful directions. The stipulation that execution of a helpful atomic action decrease the variant function guarantees O5. The stipulation that a non-helpful atomic action not increase the variant function guarantees O3. And, the stipulation that a non-helpful atomic action leave a helpful one enabled guarantees O4.

6. Discussion

Formulated in the terminology of this paper, our previous method ([Alpern & Schneider 85]) for extracting proof obligations from a deterministic Buchi automaton requires the program prover to exhibit an invariant I and a variant function v satisfying:

AS1: If x is the first state of a joint history then $x \in I$.

AS2: If $x \in I$ and $\langle x^{(\pi)},\ x'^{(\pi)}\rangle \in \alpha$ then $x' \in I$.

AS3: If $x \in I$ and $\langle x^{(\pi)},\ x'^{(\pi)}\rangle \in \alpha$
then $Pos(x) \ \vee \ Pos(x') \ \vee \ v(x') < v(x)$.

AS1 and AS2 are obligations O1 and O2 of the approach outlined in Section 4. Consider the remaining obligations (O3 through O5) of that approach. The deterministic property is a single clause that consists of a single, non-negated property, so there are no negative automata. Thus, $u(x) = \varnothing$ for every joint state x and O5 is trivially satisfied because its final disjunct is. Moreover, the final disjunct of O4 must be false. AS3 and O4 are therefore equivalent, and each implies O3. Thus, the two techniques yield essentially the same proof obligations when applied to a deterministic property.

The method in [Alpern & Schneider 85] is unsatisfactory for properties specified by non-deterministic Buchi automata. To use it to prove that a program π satisfies such a property P, a deterministic property D that is contained in P is found. Proof obligations are then extracted from the deterministic Buchi automata for D. If a (finite-state) program π satisfies P, an appropriate D always exists but may be big and difficult to find. Furthermore, the proof obligations for a non-deterministic property now depend on the program as well as on the Buchi automata for the property to be proved. The approach of Section 4 does not suffer from these difficulties. Since every property that can be specified using a non-deterministic Buchi automata can be specified as a Boolean combination of deterministic properties [Eilenberg 74] and we can now handle such Boolean combinations, our new technique is general enough to prove non-deterministic as well as deterministic properties.

In [Manna & Pnueli 87], a different technique is given for obtaining proof obligations for non-deterministic properties. The approach is based upon a ∀-automaton for a property. Simplifying slightly,[5] a ∀-automaton is a Buchi automaton that accepts its input iff every run on that input eventually is restricted to accepting states. Using the parlance of Section 4, to show that every history of a program will be accepted by a ∀-automaton M, one must exhibit an invariant I that satisfies obligations O1 and O2 and a variant function υ satisfying:

MP1: If $x \in I$, $\langle x^{(\pi)}, x'^{(\pi)} \rangle \in \alpha$, and $x'^{(1)}$ is an accepting state of M then $\upsilon(x') \leq \upsilon(x)$.

MP2: If $x \in I$, $\langle x^{(\pi)}, x'^{(\pi)} \rangle \in \alpha$, and $x'^{(1)}$ is a non-accepting state of M then $\upsilon(x') < \upsilon(x)$.

The ∀-automaton for a property is isomorphic to the Buchi automaton for the negation of that property.[6] This suggests there might be a connection between the proof obligations that are obtained from a ∀-automaton for the negation of a deterministic property and the proof obligations we obtain for a clause with a single negated property. And, there is. Since there are no positive joint states with a single negated property, O3 implies MP1, and MP1 and MP2 imply O3. O5 requires that the candidate function be empty whenever the Buchi automaton (∀-automaton) enters an accepting (non-accepting) state. Thus, we can choose the candidate function such that O4 and MP2 are equivalent—define u to be empty whenever the Buchi automata is in an accepting state and to be $\{1\}$ when it is not. Therefore, the two techniques yield the same proof obligations for a property whose negation is deterministic.

The key insight underlying the approach of [Manna & Pnueli 87] is that proof obligations for the negation of a property can be extracted directly from a Buchi automaton for the property—whether or not this automaton is deterministic. Given a property specified by a Buchi automata, to extract proof obligations using the approach of [Manna & Pnueli 87], the Buchi automata for the negation of the property is constructed. With the technique presented in this paper, the property is decomposed into a Boolean combination of properties where the non-negated terms must be specified by deterministic Buchi automata. Depending on the property, one or the other approach may be easier.

[5]We are ignoring the behavior of ∀-automata on finite sequences, the placement of transition predicates in states rather than on edges, and ∀-automata with recurrant states (which are shown to be equivalent ∀-automaton without such states in [Manna & Pnueli 87]).

[6]To obtain a ∀-automaton for P from a Buchi automaton for $\neg P$, exchange the accepting and non-accepting states.

An added advantage of the Boolean decomposition approach is that parts of the proof may be reusable since other properties might be constructed from these parts.

Acknowledgments

Discussions with Starvos Cosmadakis and Sheryl Brady were helpful, especially in formulating the completeness proof.

References

[Alpern 86] Alpern, B. Proving Temporal Properties of Concurrent Programs: A Non-Temporal Approach. Ph.D. Thesis. Department of Computer Science, Cornell University. January 1986.

[Alpern & Schneider 85] Alpern, B. and F.B. Schneider. Verifying Temporal Properties without using Temporal Logic. Submitted for publication. Also available as Technical Report TR 85-723, Department of Computer Science, Cornell University, Dec. 1985.

[Apt 81] Apt, K.R. Ten years of Hoare's logic: A survey. *ACM TOPLAS 3*, 4 (Oct. 1981), 431-483.

[Eilenberg 74] Eilenberg, S. *Automata, Languages and Machines, Vol A*. Academic Press, New York, 1974.

[Francez 86] Francez, N. *Fairness*. Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1986.

[Hoare 69] Hoare, C.A.R. An axiomatic basis for computer programming. *Commun. ACM 12*, 10 (Oct. 1969) 576-580.

[Manna & Pnueli 81] Manna, Z. and A. Pnueli. Verification of concurrent programs: The temporal framework. *The Correctness Problem in Computer Science* (R.S. Boyer and J.S. Moore, eds.), International Lecture Series in Computer Science, Academic Press, London, 1981, 141-154.

[Manna & Pnueli 84] Manna, Z. and A. Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming 4* (1984), 257-289.

[Manna & Pnueli 87] Manna, Z. and A. Pnueli. Specification and Verification of Concurrent Programs by ∀-Automata. *Proc. 14th Sym. Principles of Programming Languages*, ACM, Munich, Jan. 1987, 1-12.

[Owicki & Lamport 82] Owicki, S.S. and L. Lamport. Proving liveness properties of concurrent programs. *ACM TOPLAS 4*, 3 (July 1982), 455-496.

[Peterson 81] Peterson, G.L. Myths about the mutual exclusion problem. *Information Processing Letters 12*, 3 (June 1981), 115-116.

[Sistla et al 85] Sistla A.P., M.Y. Vardi, and P. Wolper. The Complementation Problem for Buchi Automata with Applications to Temporal Logic. *Proc. ICALP 85*, Nafplion, Greece, July 1985.

[Wolper et al 83] Wolper, P., M. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. (Extended abstract). *Proc. Twenty-fourth Annual Symposium on Foundations of Computer Science*. (Tucson, Arizona, Nov. 1983), ACM, New York, 185-194.