

Leads-to Properties and the *But-not-yet* Operator

Ken Calvert
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30032-0280
calvert@cc.gatech.edu

GIT-CC-93/60

October 1993

Abstract

We define a predicate transformer, in terms of which finite disjunctions of leads-to properties can be rewritten as single leads-to properties. Although disjunctions of leads-to properties do not typically arise naturally in progress specifications, an example shows how they may be introduced through the use of nested implications of leads-to properties; such implications allow subtle dependencies between a program's progress and that of its environment to be conveniently specified. After introducing the predicate transformer, which is called the *but-not-yet* operator, we show how to define a single leads-to property equivalent to a given disjunction of two leads-to properties. An alternative definition of the but-not-yet operator is shown to be equivalent to the first, and some properties of the operator are proved. Finally, the predicate transformer is generalized to any finite number of arguments.

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

1 Introduction

Many specification formalisms use some fragment of *linear temporal logic* to specify program properties. In such formalisms, sequences of states represent the possible behaviors of a program. A *temporal property* is a predicate on sequences of states; a program satisfies such a property P if and only if P is true for each sequence representing a possible program behaviors. In recent years it has been demonstrated that a wide range of properties useful in practice can be specified using a few simple temporal forms [4, 5, 6, 7, 8].

This paper considers a class of properties built up from basic temporal properties using boolean connectives: the temporal predicate $P \wedge Q$ is true for a sequence iff both P and Q are; $P \vee Q$ is true iff either P or Q is, and $P \Rightarrow Q$ is true iff either P is not true or Q is true. The familiar laws of propositional logic apply in the manipulation of such predicates. The basic temporal properties considered in this paper are of the form $p \rightsquigarrow q$ (“ p leads-to q ”), where p and q are predicates in the program’s variables. A sequence of states satisfies $p \rightsquigarrow q$ if each state in the sequence satisfying p is followed by a state satisfying q ; thus a program satisfies $p \rightsquigarrow q$ if, whenever it reaches a state where p holds, at that state or some later state q will hold. Leads-to properties are useful for specification of *progress*, i.e. asserting that a program *will* do something. Simple, easy-to-use proof systems, which allow elementary leads-to properties to be established for particular programs, are well known [4].

A property of the form $(x \rightsquigarrow y) \Rightarrow (p \rightsquigarrow q)$ means that every behavior of the program that satisfies $x \rightsquigarrow y$ also satisfies $p \rightsquigarrow q$. In a *compositional* theory of program specifications—one in which composition preserves all temporal properties of each component—such a property expresses the *dependence* of a program’s progress on that of its environment. (Such a theory is described, for example, in [1, 2].) In other words, $(x \rightsquigarrow y) \Rightarrow (p \rightsquigarrow q)$ is a weaker specification than $p \rightsquigarrow q$, but when a program known to satisfy that specification is placed in an environment where only behaviors satisfying $x \rightsquigarrow y$ can occur, the resulting system satisfies $p \rightsquigarrow q$. If the environment does not satisfy $x \rightsquigarrow y$, the composite of program and environment may not satisfy $p \rightsquigarrow q$; however, the original component can still be considered correct with respect to the specification $(x \rightsquigarrow y) \Rightarrow (p \rightsquigarrow q)$.

More complicated progress dependencies are sometimes useful. To see how such dependencies can arise, consider the specification of a *random bit generator* that communicates with its environment via a variable b . When the environment requests a bit by setting b to the special value \perp , the bit generator responds by setting b to either 0 or 1. *Both* 0’s and 1’s are guaranteed to be produced, but only over an infinite number of bits. In other words, there is no limit on the number of requests the environment may have to make before receiving a particular value. The conjunction of the following two implications constitutes the progress specification for the bit generator:

$$(true \rightsquigarrow b = \perp) \Rightarrow (true \rightsquigarrow b = 0) \tag{1}$$

$$(true \rightsquigarrow b = \perp) \Rightarrow (true \rightsquigarrow b = 1) \tag{2}$$

Now consider a *natural number generator* (NNG) that uses a bit generator as a subcomponent. NNG communicates with its environment via two variables: n , whose value is either a natural number or \perp , and b , through which it interacts with the bit generator. The environment sets n to \perp to request a natural number. NNG then sets b to \perp and waits for its environment (i.e. the bit generator) to set b to 0 or 1; it repeats this step until b has the value 1. Then

NNG sets n to the number of times b was 0 before it became 1. Different natural number distributions can be obtained by composing NNG with bit generators having different 0-1 distribution characteristics, so long as they satisfy the specification above.

Observe that a specification of NNG requiring $n = \perp \rightsquigarrow n \neq \perp$ is too strong. There is no way for NNG to make progress unless the bit generator does: if the latter eventually produces a 1, then NNG eventually sets n to some natural number. Moreover, requiring NNG to satisfy $true \rightsquigarrow b = \perp$ is also too strong: NNG need never change b if the environment never changes n . The correct progress specification for NNG says that if its environment satisfies the progress specification of the bit generator, NNG will eventually respond to any request:

$$((true \rightsquigarrow b = \perp) \Rightarrow (true \rightsquigarrow b = 1)) \Rightarrow (n = \perp \rightsquigarrow n \neq \perp) \quad (3)$$

Although the operational meaning of simple boolean combinations of leads-to properties is generally intuitively clear, the meaning of nested implications like (3) is not. A more serious objection to admitting such properties in specifications is that it is not clear how one would prove them for a given program.

The nested implication above is propositionally equivalent to the conjunction of the following properties:

$$(true \rightsquigarrow b = \perp) \vee (n = \perp \rightsquigarrow n \neq \perp) \quad (4)$$

$$(true \rightsquigarrow b = 1) \Rightarrow (n = \perp \rightsquigarrow n \neq \perp) \quad (5)$$

Any such nested implication can be eliminated in favor of a disjunction and a simple implication. Indeed, it can be shown that *any* positive boolean combination of elementary properties is propositionally equivalent to a property in the form of a conjunction of (simple) implications [3]. However, it is also not clear how to prove such disjunctions of leads-to properties.

It turns out that a straightforward extension to the UNITY proof rules for leads-to [4] is adequate for proving simple implications; moreover, it is not necessary to prove disjunctions. The rules for deriving and using implications will be dealt with in a separate report; the rest of this paper is devoted to showing that under a reasonable assumption about the set of sequences representing program behavior, finite disjunctions of leads-to properties can be eliminated from specifications. In particular, for any disjunction like (4) there exists a single, semantically equivalent leads-to property.

The next section introduces notation and formal definitions. Section 3 presents the main result, which asserts the existence of a specification equivalent to any disjunction of two *leads-to* properties. In order to state this result, we introduce a predicate transformer; Section 4 gives an alternative definition of this operator. Section 5 explores properties of the operator, and Section 6 generalizes the result to finite disjunctions of more than two properties. Section 7 offers some concluding remarks.

2 Definitions and Notation

Let Φ be a set of infinite sequences¹ and call the elements of these sequences *states*. The Φ represents the set of possible behaviors of some (fixed) program. Note well that the set of states under consideration contains exactly the elements that occur in some sequence in Φ ,

¹The assumption of infinite sequences is made for simplicity; the results are easily extended for the case where finite sequences are also included.

i.e. the *reachable* states of the program. The greek letters α , β , and γ will denote arbitrary sequences in Φ .

The letters i , j , and k will range over positions in sequences. For sequence α , $\alpha[i]$ denotes the i th element of α , with the initial element being $\alpha[0]$. The notation $\alpha[..i]$ denotes the finite sequence consisting of the first i elements of α , while $\alpha[i..]$ denotes the suffix remaining when the first i elements of α are removed. Note that the last element of $\alpha[..i]$ is $\alpha[i-1]$, and the first element of $\alpha[i..]$ is $\alpha[i]$.

The set Φ is postulated to have the following characteristic:

$$\text{If } \alpha \text{ and } \beta \text{ are in } \Phi, \text{ and } \alpha[i] = \beta[j], \text{ then there exists } \gamma \in \Phi \text{ such that} \quad (*) \\ \gamma[..i] = \alpha[..i] \text{ and } \gamma[i..] = \beta[j..].$$

In terms of program semantics, this can be interpreted as saying that *a program's future behavior is completely determined by its current state*.

A *state predicate* is a function from the set of states to $\{true, false\}$. The letters p , q , x and y denote state predicates; $p.s$ denotes the value of predicate p at state s . The boolean connectives are used as operators on predicates in the usual way: $(p \wedge q).s$ is equal to $p.s \wedge q.s$, etc. A state s such that $p.s = true$ is called a p -state. Universal quantification of a predicate over the entire state space is denoted by the square brackets $[]$:

$$[p] \stackrel{\text{def}}{=} \langle \forall s :: p.s = true \rangle$$

For state predicates p and q , we say a sequence α satisfies the property $p \rightsquigarrow q$ and write $\alpha \models p \rightsquigarrow q$, according to the following definition:

$$\alpha \models p \rightsquigarrow q \stackrel{\text{def}}{=} \langle \forall i :: p.\alpha[i] \Rightarrow \langle \exists j : i \leq j : q.\alpha[j] \rangle \rangle$$

Capital letters P , X , etc. denote leads-to properties. We write $\alpha \not\models P$ to mean that α does not satisfy the leads-to property P . For arbitrary leads-to properties P and Q the property $P \vee Q$ is defined by

$$\alpha \models P \vee Q \stackrel{\text{def}}{=} (\alpha \models P) \vee (\alpha \models Q)$$

We will also deal with *unless* properties; for any predicates p and q , a sequence satisfies $p \text{ untl } q$ iff every state satisfying $(p \wedge \neg q)$ is immediately followed by a state satisfying $p \vee q$.² Formally:

$$\alpha \models p \text{ untl } q \stackrel{\text{def}}{=} \langle \forall i :: (p \wedge \neg q).\alpha[i] \Rightarrow (p \vee q).\alpha[i+1] \rangle$$

We have one more notational convention regarding quantification: the double square brackets $[[]]$ stand for universal quantification over Φ : for a leads-to or *unless* property P , we have

$$[[P]] \stackrel{\text{def}}{=} \langle \forall \alpha : \alpha \in \Phi : \alpha \models P \rangle$$

For a program whose possible behaviors are described by Φ , $[[P]]$ means the program satisfies the specification P .

²The symbols used in this paper differ from those used for UNITY properties (*unless* and \mapsto), to emphasize that these are temporal predicates defined in terms of *sequences* and not properties defined in terms of the program text itself as in UNITY.

3 Eliminating Disjunctions

We would like to define, for a disjunction $P \vee Q$, a single leads-to property W such that

$$\llbracket P \vee Q \rrbracket \equiv \llbracket W \rrbracket$$

that is, a property that is equivalent to the disjunction as a specification for any program (whose behaviors satisfy $(*)$).

Clearly the form of any such W will depend upon the predicates from which P and Q are constructed. Moreover, because the desired equivalence is relative to the set Φ , the form of W may depend upon the structure of Φ as well.

The key step in constructing W will be to define a new state predicate that is true wherever the leads-to property $p \rightsquigarrow q$ is “undischarged” in a sequence—that is, a predicate true at states occurring in some sequence after a p -state has occurred but without any intervening q -state. The idea is that a sequence fails to satisfy $P \vee Q$ iff it fails to satisfy P *and* it fails to satisfy Q , i.e. if $p \rightsquigarrow q$ and $x \rightsquigarrow y$ remain undischarged forever in some infinite suffix of the sequence. We denote this predicate by $p \circledast q$ and define it as follows:³

$$(p \circledast q).s \stackrel{\text{def}}{=} \langle \exists \alpha, i, n :: \alpha \in \Phi \wedge \alpha[i] = s \wedge n \leq i \wedge p.\alpha[n] \wedge \langle \forall k : n \leq k \leq i : \neg q.\alpha[k] \rangle \rangle$$

(A similar predicate was defined by Pachl in proving completeness of the UNITY proof rules for *leads-to* [9].)

Henceforth let p, q, x and y be arbitrary fixed state predicates, and let P, X and W stand for the following leads-to properties:

$$\begin{aligned} P &\stackrel{\text{def}}{=} p \rightsquigarrow q \\ X &\stackrel{\text{def}}{=} x \rightsquigarrow y \\ W &\stackrel{\text{def}}{=} (((p \circledast q) \wedge x) \vee ((x \circledast y) \wedge p)) \rightsquigarrow (q \vee y) \end{aligned}$$

We shall establish that W is equivalent to $P \vee X$ as a specification. The first result implies that W is no weaker than $P \vee X$.

Lemma 0. For any sequence α , if $\alpha \not\models P \vee X$ then $\alpha \not\models W$.

Proof of Lemma 0. If $\alpha \not\models P \vee X$, then α satisfies neither $p \rightsquigarrow q$ nor $x \rightsquigarrow y$, and in particular there exist states $\alpha[i]$ and $\alpha[j]$ such that

$$p.\alpha[i] \wedge \langle \forall k : i \leq k : \neg q.\alpha[k] \rangle \tag{6}$$

$$x.\alpha[j] \wedge \langle \forall k : j \leq k : \neg y.\alpha[k] \rangle \tag{7}$$

Now suppose $i \leq j$. We observe:

$$\begin{aligned} &i \leq j \wedge (6) \wedge (7) \\ = &\{ \text{predicate calculus—split range in (6)} \} \\ &i \leq j \wedge p.\alpha[i] \wedge \langle \forall k : i \leq k \leq j : \neg q.\alpha[k] \rangle \wedge \end{aligned}$$

³Note that \circledast does depend on Φ ; because Φ is assumed fixed, this dependence will be left implicit.

$$\begin{aligned}
& x.\alpha[j] \wedge \langle \forall k : j \leq k : \neg q.\alpha[k] \rangle \wedge \langle \forall k : j \leq k : \neg y.\alpha[k] \rangle \\
\Rightarrow & \quad \{ \text{existential generalization and predicate calculus} \} \\
& \langle \exists n : n \leq j : p.\alpha[n] \wedge \langle \forall k : n \leq k \leq j : \neg q.\alpha[k] \rangle \rangle \wedge \\
& \quad x.\alpha[j] \wedge \langle \forall k : j \leq k : (\neg q \wedge \neg y).\alpha[k] \rangle \\
\Rightarrow & \quad \{ \text{definition of } p \circledast q; \text{ DeMorgan's law} \} \\
& (p \circledast q).\alpha[j] \wedge x.\alpha[j] \wedge \langle \forall k : j \leq k : \neg(q \vee y).\alpha[k] \rangle \\
= & \quad \{ \text{predicate calculus} \} \\
& ((p \circledast q) \wedge x).\alpha[j] \wedge \langle \forall k : j \leq k : \neg(q \vee y).\alpha[k] \rangle \\
\Rightarrow & \quad \{ \text{weakening the first conjunct} \} \\
& (((p \circledast q) \wedge x) \vee ((x \circledast y) \wedge p)).\alpha[j] \wedge \langle \forall k : j \leq k : \neg(q \vee y).\alpha[k] \rangle \\
\Rightarrow & \quad \{ \text{definition of } W \} \\
& \alpha \not\models W
\end{aligned}$$

For the case $j \leq i$, a similar argument again proves $\alpha \not\models W$. \square

Lemma 1. If there exists a sequence $\beta \in \Phi$ and a state $\beta[j]$ such that

$$((p \circledast q) \wedge x).\beta[j] \wedge \langle \forall k : j \leq k : \neg(q \vee y).\beta[k] \rangle$$

then there exists a sequence $\gamma \in \Phi$ such that $\gamma \not\models (p \rightsquigarrow q) \vee (x \rightsquigarrow y)$.

Proof of Lemma 1. Let β and j satisfy the hypothesis of the Lemma. Because $\beta[j]$ is a $p \circledast q$ -state, there exists a sequence α in Φ and i and n such that $\alpha[i] = \beta[j]$, and

$$n \leq i \wedge p.\alpha[n] \wedge \langle \forall k : n \leq k \leq i : \neg q.\alpha[k] \rangle \quad (8)$$

Because α and β are in Φ and $\alpha[i] = \beta[j]$, by property (*) there exists a sequence γ in Φ such that $\gamma[..*i*] = \alpha[..*i*]$ and $\gamma[..*j*] = \beta[..*j*]$. From this fact, the hypothesis, and (8) we have the following properties of γ :

$$x.\gamma[i] \quad (9)$$

$$\langle \forall k : i \leq k : \neg(q \vee y).\gamma[k] \rangle \quad (10)$$

$$p.\gamma[n] \wedge \langle \forall k : n \leq k \leq i : \neg q.\gamma[k] \rangle \quad (11)$$

Now we observe:

$$\begin{aligned}
& (10) \wedge (11) \\
\Rightarrow & \quad \{ \text{predicate calculus} \} \\
& p.\gamma[n] \wedge \langle \forall k : n \leq k : \neg q.\gamma[k] \rangle \\
= & \quad \{ \text{definition} \} \\
& \gamma \not\models p \rightsquigarrow q
\end{aligned}$$

The symmetric argument establishes $\gamma \not\models x \rightsquigarrow y$, and thus $\gamma \not\models (p \rightsquigarrow q) \vee (x \rightsquigarrow y)$. \square

Now by a similar argument we can prove

Lemma 2. If there exists a sequence β in Φ , and a position j in β such that

$$((x \circledast y) \wedge p).\beta[j] \wedge \langle \forall k : j \leq k : \neg(q \vee y).\beta[k] \rangle$$

then there exists a sequence $\gamma \in \Phi$ such that $\gamma \not\models (p \rightsquigarrow q) \vee (x \rightsquigarrow y)$.

Using the foregoing Lemmata, we can show that W is no stronger than $P \vee X$.

Lemma 3. $\llbracket P \vee X \rrbracket \Rightarrow \llbracket W \rrbracket$.

Proof of Lemma 3. We prove the contrapositive, i.e. if there exists a sequence $\beta \in \Phi$ such that $\beta \not\models W$, then there exists a sequence $\gamma \in \Phi$ such that $\gamma \not\models P \vee X$. Let β be any sequence such that $\beta \not\models W$. By definition, there exists a state $\beta[j]$ such that

$$(((p \otimes q) \wedge x) \vee ((x \otimes y) \wedge p)).\beta[j] \wedge \langle \forall k : j \leq k : \neg(q \vee y).\beta[k] \rangle$$

which is equivalent to the disjunction of the following:

$$\begin{aligned} & ((p \otimes q) \wedge x).\beta[j] \wedge \langle \forall k : j \leq k : \neg(q \vee y).\beta[k] \rangle \\ & ((x \otimes y) \wedge p).\beta[j] \wedge \langle \forall k : j \leq k : \neg(q \vee y).\beta[k] \rangle \end{aligned}$$

Thus $\beta[j]$ satisfies the hypothesis of either Lemma 1 or Lemma 2. The result follows from the definitions of P and X and the appropriate Lemma. \square

Theorem 0. For fixed Φ with property (*), and P , X and W as defined above:

$$\llbracket P \vee X \rrbracket \equiv \llbracket W \rrbracket$$

Proof of Theorem 0. The right side implies the left by the contrapositive of Lemma 0 and the monotonicity of universal quantification; the left side implies the right by Lemma 3. \square

For the example given in the first section, we can replace property (4) in the specification by

$$((\text{true} \otimes b = \perp) \wedge n = \perp) \vee (n = \perp \otimes n \neq \perp) \rightsquigarrow (b = 1 \vee n \neq \perp) \quad (12)$$

4 Alternative Definition of $p \otimes q$

The definition of the predicate $p \otimes q$ given above is tailored to the proof of Theorem 0. In practice, however, we want a definition that is conveniently representable. Therefore we give an alternative characterization of the predicate using *unless* properties.

For given predicates p and q , consider predicates z satisfying the following:

$$\llbracket (p \wedge \neg q) \Rightarrow z \rrbracket \quad (13)$$

$$\llbracket z \text{ unl } q \rrbracket \quad (14)$$

Henceforth we write $\text{BNY}(p, q, z)$ to indicate that predicates p , q and z satisfies (13) and (14); we write $\text{BNY}(x, y, w)$ to indicate that x , y and w satisfy

$$\begin{aligned} & \llbracket (x \wedge \neg y) \Rightarrow w \rrbracket \\ & \llbracket w \text{ unl } y \rrbracket \end{aligned}$$

For any p and q , there is at least one predicate z such that $\text{BNY}(p, q, z)$: *true* satisfies the conditions trivially. The following theorem states that $p \otimes q$ can be characterized as an extreme solution to the “equation”

$$z : \text{BNY}(p, q, z).$$

Theorem 1. The predicate $p \otimes q$ is the strongest predicate z satisfying (13) and (14), that is:

$$[(p \wedge \neg q) \Rightarrow (p \otimes q)] \quad (15)$$

$$\llbracket p \otimes q \text{ unl } q \rrbracket \quad (16)$$

$$\langle \forall z : \text{BNY}(p, q, z) : [(p \otimes q) \Rightarrow z] \rangle \quad (17)$$

Proof of (15). We observe for any state s :

$$\begin{aligned} & (p \wedge \neg q).s \\ = & \{ \text{every state occurs in some sequence in } \Phi \} \\ & \langle \exists \alpha, j :: \alpha[j] = s \wedge (p \wedge \neg q).\alpha[j] \rangle \\ = & \{ \text{predicate calculus} \} \\ & \langle \exists \alpha, j :: \alpha[j] = s \wedge j \leq j \wedge p.\alpha[j] \wedge \langle \forall k : j \leq k \leq j : \neg q.\alpha[k] \rangle \rangle \\ \Rightarrow & \{ \text{existential instantiation} \} \\ & \langle \exists \alpha, j, n :: \alpha[j] = s \wedge n \leq j \wedge p.\alpha[n] \wedge \langle \forall k : n \leq k \leq j : \neg q.\alpha[k] \rangle \rangle \\ = & \{ \text{definition} \} \\ & (p \otimes q).s \end{aligned}$$

□

Proof of (16). Our proof obligation is:

$$\langle \forall \beta, j :: ((p \otimes q) \wedge \neg q).\beta[j] \Rightarrow ((p \otimes q) \vee q).\beta[j + 1] \rangle$$

The term of this formula can be rewritten:

$$\begin{aligned} & ((p \otimes q) \wedge \neg q).\beta[j] \Rightarrow ((p \otimes q) \vee q).\beta[j + 1] \\ = & \{ [p \otimes q \Rightarrow \neg q] \text{ by definition; predicate calculus} \} \\ & (p \otimes q).\beta[j] \Rightarrow (p \otimes q).\beta[j + 1] \vee q.\beta[j + 1] \\ = & \{ \text{predicate calculus} \} \\ & (p \otimes q).\beta[j] \wedge \neg q.\beta[j + 1] \Rightarrow (p \otimes q).\beta[j + 1] \end{aligned}$$

Now let β and j be such that $(p \otimes q).\beta[j]$ and $\neg q.\beta[j + 1]$. We shall establish $(p \otimes q).\beta[j + 1]$. According to the definition of $p \otimes q$, there exists a sequence α , and n and i such that

$$\alpha[i] = \beta[j] \wedge n \leq i \wedge p.\alpha[n] \wedge \langle \forall k : n \leq k \leq i : \neg q.\alpha[k] \rangle$$

Thanks to property (*) and the above, there exists a sequence γ such that $\beta[j..] = \gamma[i..]$; from the properties of α and β , we have the following property of this sequence:

$$n \leq i \wedge p.\gamma[n] \wedge \langle \forall k : n \leq k \leq i : \neg q.\gamma[k] \rangle \wedge \neg q.\gamma[i + 1] \quad (18)$$

Now we calculate:

$$\begin{aligned} & (18) \\ \Rightarrow & \{ \text{predicate calculus} \} \\ & n \leq i + 1 \wedge p.\gamma[n] \wedge \langle \forall k : n \leq k \leq i + 1 : \neg q.\gamma[k] \rangle \\ \Rightarrow & \{ \text{definition} \} \\ & (p \otimes q).\gamma[i + 1] \\ = & \{ \gamma[i..] = \beta[j..] \} \\ & (p \otimes q).\beta[j + 1] \end{aligned}$$

□

Proof of (17). Let z be any predicate such that $\text{BNY}(p, q, z)$, and let s be any state such that $(p \circledast q).s$; we shall establish $z.s$.

By hypothesis there exist α, j and $n \leq j$ such that $\alpha[j] = s$ and

$$p.\alpha[n] \wedge \langle \forall k : n \leq k \leq j : \neg q.\alpha[k] \rangle \quad (19)$$

From this follows $(p \wedge \neg q).\alpha[n]$, which implies (because z satisfies (13))

$$z.\alpha[n] \quad (20)$$

In case $n = j$, we have established $z.\alpha[j]$ as required. Otherwise, we have $n < j$. For this case, we shall establish that for each k in the range $n \leq k < j$,

$$z.\alpha[k] \Rightarrow z.\alpha[k+1]. \quad (21)$$

From this and (20), $z.\alpha[j]$ follows by induction.

Because z satisfies (14) we have for this α and any k' ,

$$(z \wedge \neg q).\alpha[k'] \Rightarrow (z \vee q).\alpha[k'+1] \quad (22)$$

Now assume that $z.\alpha[k]$ holds for $n \leq k < j$. We observe:

$$\begin{aligned} & z.\alpha[k] \\ = & \{ k \text{ is in the range of the quantification in (19)} \} \\ & (z \wedge \neg q).\alpha[k] \\ \Rightarrow & \{ (22) \} \\ & (z \vee q).\alpha[k+1] \\ = & \{ k < j, \text{ so } k+1 \text{ is in the range of quantification in (19)} \} \\ & ((z \vee q) \wedge \neg q).\alpha[k+1] \\ \Rightarrow & \{ \text{predicate calculus} \} \\ & z.\alpha[k+1] \end{aligned}$$

Thus we have established (21), and hence $z.\alpha[j]$, that is, $z.s$. □

As a consequence of these results, if we want to prove that every sequence in Φ satisfies W , it is sufficient to find a predicates z and w such that $\text{BNY}(p, q, z)$ and $\text{BNY}(x, y, w)$, and such that the following leads-to property is provable:

$$(z \wedge x) \vee (w \wedge p) \rightsquigarrow (q \vee y).$$

By the foregoing, we have $p \circledast q \Rightarrow z$ and $x \circledast y \Rightarrow w$ for any such z and w ; the leads-to property W then follows by the operational definition of leads-to.

5 Properties of \circledast

In this section we briefly explore some of the simple properties of \circledast as a predicate transformer, using only the “definition” given in Theorem 1.

We begin with some simple identities.

$$[true \circledast q \equiv \neg q] \tag{23}$$

Proof of (23). $\text{BNY}(true, q, \neg q)$ holds trivially, and thus (by Theorem 1) we have $[true \circledast q \Rightarrow \neg q]$. But by the same result we have also $\text{BNY}(true, q, true \circledast q)$, and thus $[true \wedge \neg q \Rightarrow true \circledast q]$ \square

By similar arguments, we can prove

$$[\neg q \circledast q \equiv \neg q] \tag{24}$$

$$[p \circledast true \equiv false] \tag{25}$$

Proof of (25). From Theorem 1 $p \circledast true$ is the strongest z such that $\text{BNY}(p, true, z)$, i.e. such that $[false \Rightarrow z]$ and $\llbracket z \text{ unl } true \rrbracket$. Evidently $false$ satisfies both of these, and is the strongest predicate to do so. \square

A similar argument proves

$$[false \circledast q \equiv false] \tag{26}$$

The predicate $p \circledast false$ corresponds to the strongest stable predicate weaker than p , which has been named (in the context of UNITY) *sst.p* by Sanders [10].

We turn now to monotonicity properties. The first of these is that \circledast is monotonic in its first argument, i.e.

$$[x \Rightarrow y] \Rightarrow [x \circledast q \Rightarrow y \circledast q] \tag{27}$$

Proof of (27). It is sufficient to show $\text{BNY}(x, q, y \circledast q)$ using $[x \Rightarrow y]$; the result then follows by Theorem 1. Our proof obligations are $[x \wedge \neg q \Rightarrow y \circledast q]$ and $\llbracket (y \circledast q) \text{ unl } q \rrbracket$. The latter property holds, by Theorem 1 For the former, we observe:

$$\begin{aligned} & x \wedge \neg q \\ \Rightarrow & \{ [x \Rightarrow y] \} \\ & y \wedge \neg q \\ \Rightarrow & \{ \text{BNY}(y, q, y \circledast q), \text{Theorem 1} \} \\ & y \circledast q \end{aligned}$$

\square

With respect to its second argument, \circledast is antimonotonic, i.e.

$$[x \Rightarrow y] \Rightarrow [p \circledast y \Rightarrow p \circledast x] \tag{28}$$

Proof of (28). Again, it is sufficient to show $\text{BNY}(p, y, p \circledast x)$. Our first obligation is $[p \wedge \neg y \Rightarrow p \circledast x]$; we observe:

$$\begin{aligned}
& p \wedge \neg y \\
\Rightarrow & \{ [x \Rightarrow y] \} \\
& p \wedge \neg x \\
\Rightarrow & \{ \text{BNY}(p, x, p \circ x) \text{ and Theorem 1} \} \\
& p \circ x
\end{aligned}$$

Our second proof obligation is $\llbracket p \circ x \text{ unl } y \rrbracket$. But this follows because *unl* is monotonic in its second argument:

$$[x \Rightarrow y] \Rightarrow (\llbracket p \text{ unl } x \rrbracket \Rightarrow \llbracket p \text{ unl } y \rrbracket)$$

and we have $\llbracket p \text{ unl } x \rrbracket$ by Theorem 1. \square

As for junctivity properties of \circ , it is universally disjunctive in its first argument, i.e. for any set W such that $x.i$ is a predicate for each $i \in W$:

$$[\langle \exists i : i \in W : x.i \rangle \circ q] \equiv \langle \exists i : i \in W : x.i \circ q \rangle \quad (29)$$

However, \circ enjoys no other interesting junctivity properties. It is not conjunctive in its first argument, because $(x \circ q) \wedge (y \circ q)$ may hold at a state that is not reachable via any sequence that goes through a state where $x \wedge y$ holds—for example if $y = \neg x$. By considering that $p \circ (x \wedge y)$ may hold at a state where x holds, and therefore $p \circ x$ does *not* hold, we can see that it is not conjunctive in its second argument either.

That \circ is not disjunctive in its second argument can be seen operationally by considering that if $p \circ (x \vee y)$ holds at a state, neither x nor y holds at that state; the latter is not necessarily true of a state satisfying $(p \circ x) \vee (p \circ y)$. However, we do have the following:

$$[p \circ (x \vee y)] \Rightarrow (p \circ x) \wedge (p \circ y) \quad (30)$$

Finally, we observe that if $\llbracket p \text{ unl } q \rrbracket$ holds, we have also (by properties of *unl*) $\llbracket p \wedge \neg q \text{ unl } q \rrbracket$, and thus $\text{BNY}(p, q, p \wedge \neg q)$; because $p \wedge \neg q$ is the obviously the strongest predicate for which this can be true, we have

$$[(p \circ q) \equiv (p \wedge \neg q)]$$

For the example given in the first section, every sequence trivially satisfies both *true unl* $b = 1$ and $n = \perp \text{ unl } n \neq \perp$; thus *true* $\circ b = 1$ is equivalent to $b \neq 1$ and $n = \perp \circ n \neq \perp \equiv n = \perp$. Property (12) thus becomes:

$$(b \neq 1 \wedge n = \perp) \rightsquigarrow b = 1 \vee n \neq \perp$$

6 Generalization to Finite Disjunctions

Theorem 0 can be generalized to disjunctions of more than two properties. Doing so requires that the operator \circ be generalized to define a predicate true at any state where each of a collection of leads-to properties is undischarged. That is, to eliminate a finite disjunction of N leads-to properties

$$\langle \vee m : 0 \leq m < N : p_m \rightsquigarrow q_m \rangle$$

we have to define a predicate that is true at any state in a sequence where *each* p_m has occurred earlier in the sequence, and there is no intervening q_m -state. To this end, let A be

the set of predicate pairs (p_m, q_m) from the above disjunction. Define the predicate $\Gamma.A$ as follows:

$$(\Gamma.A).s \stackrel{\text{def}}{=} \langle \exists \alpha, k : \alpha \in \Phi \wedge \alpha[k] = s : \\ \langle \forall m :: \langle \exists i : i \leq k : p_m.\alpha[i] \wedge \langle \forall j : i \leq j \leq k : \neg q_m.\alpha[j] \rangle \rangle \rangle \rangle$$

We then have:

Theorem 2. For $N > 0$, let A be the set of predicate pairs (p_m, q_m) , $m = 0, 1, \dots, N$. Let $\Gamma.A$ be the predicate defined above. For any set Φ of sequences with property (*):

$$\langle \forall \alpha : \alpha \in \Phi : \alpha \models p_0 \rightsquigarrow q_0 \vee \dots \vee p_N \rightsquigarrow q_N \rangle \equiv \\ \langle \forall \alpha : \alpha \in \Phi : \alpha \models \Gamma.A \rightsquigarrow q_0 \vee \dots \vee q_N \rangle$$

A proof of Theorem 2, along with additional details, may be found in [3].

7 Discussion

The results presented here can be used to show that complex dependencies among progress properties can be represented by specifications expressed using only conjunctions and simple implications of leads-to properties. The advantage of stating specifications in this restricted form is that only a few small extensions to the simple and elegant proof rules for leads-to (e.g., as in [4]) are required to obtain a complete proof system for such properties.

Acknowledgement

This paper has benefitted from helpful comments by J. R. Rao and Ted Herman.

References

- [1] Kenneth L. Calvert. Module composition and refinement: Extending the lam-shankar theory. Technical Report GIT-CC-91/58, College of Computing, Georgia Institute of Technology, 1991.
- [2] Kenneth L. Calvert. Module composition and refinement with applications to protocol conversion. In *Proceedings XII Symposium on Protocol Specification, Testing, and Verification, Orlando, Florida*. North-Holland, June 1992.
- [3] Kenneth L. Calvert. Specifying progress properties with leads-to. Technical Report GIT-CC-92/59, College of Computing, Georgia Institute of Technology, December 1992.
- [4] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.
- [5] Simon S. Lam and A. Udaya Shankar. A relational notation for state transition systems. *IEEE Transactions on Software Engineering*, 16(7):755–775, July 1990.

- [6] Leslie Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, 1983.
- [7] Leslie Lamport. A temporal logic of actions. Technical Report Research Report 57, DEC Systems Research Center, April 1990.
- [8] Zohar Manna and Amir Pnueli. *Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
- [9] Jan Pachl. A simple proof of a completeness result for *leads-to* in the unity logic. *Information Processing Letters*, 41:35–38, 1992.
- [10] Beverly A. Sanders. A predicate transformer approach to knowledge and knowledge-based protocols. Technical Report 181, Eidgenössische Technische Hochschule Zürich, Institut für Computersysteme, September 1992.