Conjunctive Predicate Transformers for Reasoning about Concurrent Computation

K. Mani Chandy

California Institute of Technology 256-80
Pasadena, California 91125
U.S.A.

Beverly A. Sanders*

Institute für Computersysteme
Swiss Federal Institute of Technology (ETH Zürich)
ETH Zentrum
CH-8092 Zürich
Switzerland

June 18, 1993

Abstract

In this paper we propose a calculus for reasoning about concurrent programs inspired by the *wp* calculus for reasoning about sequential programs. We suggest predicate transformers for reasoning about progress properties and for deducing properties obtained by parallel composition. The paper presents theorems about the predicate transformers and suggests how they can be used in program design. Familiarity with the *wp* calculus is assumed.

1 Definitions

A *program* is a finite set of typed variables and a finite, nonempty set of commands. The state of a program is given by the values of its variables. The program state is changed by executing a command, and *wp.s* is monotonic, universally-conjunctive and or-continuous [?]. A *computation* of a program is

*Supported in part by Swiss National Science Foundation grant 5003-034260.

an initial state S_0, and a sequence of pairs $(c_i, S_i), i > 0$, where c_i is a command and S_i is a program state, and execution of command c_i can take the program from state S_{i-1} to state S_i, for $i > 0$, and each command in the program appears infinitely often.

The *parallel composition* of programs F and G is defined if and only if the variable declarations in F and G are compatible, and is denoted by $F||G$. The set of variables of $F||G$ is the union of the sets of variables of F and G. Likewise, the set of commands of $F||G$ is the union of the sets of commands of F and G. A variable appearing in both F and G is shared by both programs.

A program can be denoted as a UNITY program without *initially* or *always* sections [?]. A program can also be a TLA formula with an initial condition containing only variable type declarations [?]. For the restricted purposes of this paper, the choice of notation is unimportant.

A *program property* is a predicate on programs. A program property which is helpful in reasoning about progress is $\leadsto$ ("leads to"), where $p \leadsto q$ holds for a program F if and only if in all infinite computations of F, if p holds at any point in the computation then q holds at that point or a later point [?].

We postulate an equivalence operator on programs. We deliberately do not define program equivalence formally because it is not necessary for the purposes of this paper. ¿From the point of view of program derivation, we define a restricted set of program properties that we use to specify and reason about programs, and require that if two programs are equivalent then at least every property (from this restricted set) of one program is also a property of the other.

In the paper we use p, q, and r for predicates on states; U, V, and W for properties of programs; and F and G for programs.

2 Safety properties

For a program F we define a predicate transformer $awp.F$ (which stands for *a*ll *wp*) defined as:

$$awp.F.q \triangleq (\forall s : s \text{ is a command of } F : wp.s.q) \tag{1}$$

¿From the meaning of wp it follows that $awp.F.q$ is the weakest predicate p such that executing any command of F in a state that satisfies p will terminate in a state that satisfies q.

Since $wp.s$ is monotonic, universally conjunctive and or-continuous, for each command s, we conclude that $awp.F$ is also monotonic, universally conjunctive and or-continuous. It is neither idempotent nor finitely disjunctive.[1]

If the parallel composition of F and G is defined, then from (1):

$$[awp.F||G.q \equiv awp.F.q \wedge awp.G.q]. \tag{2}$$

[1]Let s_1 = "$x := x + 1$", s_2 = "$x := x + 2$", $q = (x = 2), q\prime = (x = 3)$. Then, $awp.q = false, awp.q\prime = false, awp.q \vee q\prime = (x = 1)$.

For a predicate p on program states, we define a program property *stable* p as follows.

$$(\text{stable } p).F \;\; \triangleq \;\; [p \Rightarrow awp.F.p] \tag{3}$$

Therefore, *stable* p holds for a program F if and only if all states of F reachable from states that satisfy p also satisfy p.

¿From (2) and (3), if $F||G$ is defined:

$$(\text{stable } p).F||G \;\equiv\; (\text{stable } p).F \wedge (\text{stable } p).G$$

¿From the definition of stability (3), for any set of predicates $\{p_w | w \in W\}$:

$$(\forall w : w \in W : \text{stable } p_w) \Rightarrow \text{stable } (\exists w : w \in W : p_w) \tag{4}$$

$$(\forall w : w \in W : \text{stable } p_w) \Rightarrow \text{stable } (\forall w : w \in W : p_w)) \tag{5}$$

For any predicate q, we define $wst.q$ [?, ?] to be the weakest predicate stronger than q which is stable in F:

$$wst.F.q \;\; \triangleq \;\; \text{weakest } p : \; [p \;\Rightarrow\; awp.F.p \wedge q]$$

Operationally, if $wst.F.q$ holds at any point in a computation of F then q holds at that point and for ever thereafter.

¿From the definition, it follows that

$$(\text{ stable } q).F \equiv [wst.F.q \equiv q] \tag{6}$$

$$(\text{ stable } wst.F.q).F \tag{7}$$

¿From the Knaster-Tarski theorem [?] and the monotonicity of awp

$$[wst.F.q \;\;\equiv\;\; \text{weakest } p : [p \equiv awp.F.p \wedge q]]$$

¿From [?], wst inherits monotonicity, universal conjunctivity, and or-continuity from awp. Since awp is universally conjunctive, it is and-continuous and therefore:

$$[wst.F.q \equiv (\forall i : i \geq 0 : f^i.true)] \text{ where } f.x = awp.F.x \wedge q$$

Thus, for finite state programs, wst can be calculated from awp.

3 Progress Properties

3.1 Leads-to and To-always

Consider the program F with the single integer variable x and a single command $x := x + 1$. Then

$$((x = 0) \rightsquigarrow (x = 1)).F$$

and

$$((x = 0) \rightsquigarrow (x = 2)).F$$

but

$$\neg(x = 0 \rightsquigarrow (x = 1 \wedge x = 2)).F$$

Conjunctivity is helpful, and so we explore (finitely) conjunctive predicate transformers that help in reasoning about progress.

Knapp [?] proposed a predicate transformer *wlt*, for weakest leads to, defined as:

$$p \rightsquigarrow q \text{ in } F \ \triangleq\ [p \Rightarrow wlt.F.q] \tag{8}$$

where p and q are predicates on states. Knapp showed that *wlt* is monotonic and idempotent but neither finitely conjunctive, finitely disjunctive, and-continuous nor or-continuous.

We define a predicate transformer $wto.F$ on predicates of states of F as:

$$wto.F.q \triangleq wlt.F.(wst.F.q) \tag{9}$$

If $wto.F.q$ holds at any point t in an infinite computation of F, then there is a point t' in the computation at which ($wst.F.q$ holds, and hence) q holds and continues to hold for ever thereafter.

Let us define a program property, $p \hookrightarrow q$, ("to always") where p and q are predicates on program states, defined as:

$$(p \hookrightarrow q).F \ \triangleq\ [p \Rightarrow wto.F.q] \tag{10}$$

Then if $(p \hookrightarrow q).F$, and there is a point in an infinite computation of F at which p holds, then there is a point in the computation after which q continues to hold for ever [?].

Since *wlt* and *wst* are monotonic, so is *wto*. We shall prove that *wto* is idempotent and finitely conjunctive. We observe that *wto* is neither finitely disjunctive nor or-continuous.

In the following, for convenience, we shall drop references to the program under consideration, and we shall use "$wlt.q$" in place of "$wlt.F.q$," with the understanding that all such statements refer to the same program.

The following three theorems from [?] and [?] are used in the proofs.

$$[q \Rightarrow wlt.q] \tag{11}$$

$$[(\neg q \wedge wlt.q) \;\Rightarrow\; awp.wlt.q] \tag{12}$$

$$\text{stable } r \;\Rightarrow\; [(wlt.q \wedge r) \;\Rightarrow\; wlt.(q \wedge r)] \tag{13}$$

Theorem

$$\text{stable } q \;\Rightarrow\; \text{stable } wlt.q \tag{14}$$

Proof:

$$
\begin{array}{ll}
 & \text{stable } q \\
\equiv & \qquad \{ \text{ definition of stable, (3) } \} \\
 & [q \Rightarrow awp.q] \\
\Rightarrow & \qquad \{ \text{ disjunction with (12) } \} \\
 & [q \vee (\neg q \wedge wlt.q) \;\Rightarrow\; awp.q \vee awp.wlt.q] \\
\equiv & \qquad \{ \text{ predicate calculus } \} \\
 & [q \vee wlt.q \;\Rightarrow\; awp.q \vee awp.wlt.q] \\
\equiv & \qquad \{ \text{ strengthen antecedent } \} \\
 & [wlt.q \;\Rightarrow\; awp.q \vee awp.wlt.q] \\
\Rightarrow & \qquad \{ \; (awp.a \vee awp.b) \Rightarrow awp.(a \vee b), \text{ from monotonicity of } awp \; \} \\
 & [wlt.q \;\Rightarrow\; awp.(q \vee wlt.q)] \\
\equiv & \qquad \{ \; q \vee wlt.q \equiv wlt.q, \text{ from (11) } \} \\
 & [wlt.q \;\Rightarrow\; awp.wlt.q] \\
\equiv & \qquad \{ \text{ definition of stable, (3) } \} \\
 & \text{stable } wlt.q
\end{array}
$$

Corollary

$$\text{stable } wto.q \tag{15}$$

Proof

$$
\begin{array}{ll}
 & \text{true} \\
\equiv & \qquad \{ \text{ (7) } \} \\
 & \text{stable } wst.q \\
\Rightarrow & \qquad \{ \text{ (14), with } q := wst.q \; \} \\
 & \text{stable } wlt.wst.q \\
\equiv & \qquad \{ \text{ definition of } wto \; \} \\
 & \text{stable } wto.q
\end{array}
$$

Theorem (wto idempotent)

$$[wto.(wto.q) \equiv wto.q]$$

Proof:

$wto.(wto.q)$
$\equiv$ $\{$ definition of wto $\}$
$wlt.(wst.(wto.q))$
$\equiv$ $\{wst.wto.q = wto.q$, from (15), (6) $\}$
$wlt.(wto.q)$
$\equiv$ $\{$ definition of wto, (9) $\}$
$wlt.(wlt.wst.q)$
$\equiv$ $\{$ wlt idempotent $\}$
$wlt.wst.q$
$\equiv$ $\{$ definition of wto $\}$
$wto.q$

Theorem (wto finitely conjunctive)

$$[wto.q \wedge wto.q' \equiv wto.(q \wedge q')]$$

Proof:

$wto.q \wedge wto.q'$
$\equiv$ $\{$ definition of wto $\}$
$wlt.wst.q \wedge wto.q'$
$\Rightarrow$ $\{$ (13),(15) $\}$
$wlt.(wst.q \wedge wto.q')$
$\equiv$ $\{$ definition of wto $\}$
$wlt.(wst.q \wedge wlt.wst.q')$
$\Rightarrow$ $\{$ (13), stable $wst.q$ (7), with $r := wst.q$, $q := wst.q'$ $\}$
$wlt.wlt.(wst.q \wedge wst.q')$
$\equiv$ $\{$ wlt idempotent $\}$
$wlt.(wst.q \wedge wst.q')$
$\equiv$ $\{$ wst conjunctive $\}$
$wlt.wst.(q \wedge q')$
$\equiv$ $\{$ definition of wto $\}$
$wto.(q \wedge q')$
$\equiv$ $\{$ predicate calculus $\}$
$wto.(q \wedge q') \wedge wto.(q \wedge q')$
$\Rightarrow$ $\{$ wto monotonic, $(q \wedge q') \Rightarrow q$, $(q \wedge q') \Rightarrow q'$ $\}$
$wto.q \wedge wto.q'$

3.2 Proofs of $\hookrightarrow$ properties

For finite state programs, wto can be calculated, at least in principle. However, weaker rules that allow the conclusion of to-always properties are useful in practice. The definition of wto in terms of wlt and wst allows known results for leads-to to be applied.

¿From [?], define the property p co q,[2] where p and q are predicates on program states as:

$$(p \text{ co } q).F \;\equiv\; [p \Rightarrow q] \wedge [p \Rightarrow awp.F.q]$$

Define the property $E(p,q)$ as:

$$E(p,q).F \triangleq (\exists s : s \text{ is a command in } F \;: [p \wedge \neg q \;\Rightarrow wp.s.q])$$

Then, the following is known to be a sound proof rule for leads-to:

$$[(p \wedge \neg q \text{ co } p \vee q) \wedge E(p,q) \Rightarrow p \leadsto q] \tag{16}$$

As a consequence of this and (6), the following is a sound proof rule for to-always:

$$[(p \wedge \neg q \text{ co } p \vee q) \wedge E(p,q) \wedge \text{ stable } q \Rightarrow p \hookrightarrow q] \tag{17}$$

Additional useful metatheorems are easily obtained from the properties of wto.

Theorem ($\hookrightarrow$ is transitive)

$$(p \hookrightarrow q \wedge q \hookrightarrow r) \Rightarrow p \hookrightarrow r \tag{18}$$

Proof:

$$
\begin{array}{ll}
 & p \hookrightarrow q \wedge q \hookrightarrow r \\
\equiv & \qquad \{\text{definition of } \hookrightarrow\} \\
 & [p \Rightarrow wto.q] \wedge [q \Rightarrow wto.r] \\
\Rightarrow & \qquad \{wto \text{ monotonic}\} \\
 & [p \Rightarrow wto.(wto.r)] \\
\equiv & \qquad \{wto \text{ idempotent}\} \\
 & [p \Rightarrow wto.r] \\
\equiv & \qquad \{\text{definition of } \hookrightarrow\} \\
 & p \hookrightarrow r
\end{array}
$$

[2] The familiar UNITY property p unless q is equivalent to $p \wedge \neg q$ co $p \vee q$.

Theorem

$$\text{stable } p \Rightarrow (p \hookrightarrow p) \tag{19}$$

Proof:

$$\begin{array}{ll}
 & \text{stable } p \\
\equiv & \qquad \{(6)\} \\
 & [p \equiv wst.p] \\
\equiv & \qquad \{(11),\ q := wst.p\ \} \\
 & [(p \equiv wst.p) \wedge (wst.p \Rightarrow wlt.wst.p)] \\
\Rightarrow & \qquad \{\text{ predicate calculus }\} \\
 & [p \Rightarrow wlt.wst.p] \\
\equiv & \qquad \{\text{ definition of } wto\ \} \\
 & [p \Rightarrow wto.p] \\
\equiv & \qquad \{\text{ definition of } \hookrightarrow\ \} \\
 & p \hookrightarrow p
\end{array}$$

Theorem ($\hookrightarrow$ universally disjunctive)
For arbitrary set W:

$$(\forall w : w \in W : p_w \hookrightarrow q_w) \Rightarrow ((\exists w : w \in W : p_w) \hookrightarrow (\exists w : w \in W : q_w)) \tag{20}$$

Theorem ($\hookrightarrow$ finitely conjunctive) For finite set W:

$$[(\forall w : w \in W : p_w \hookrightarrow q_w) \Rightarrow (\forall w : w \in W : p_w) \hookrightarrow (\forall w : w \in W : q_w)] \tag{21}$$

Theorem (strengthen left side, weaken right side of $\hookrightarrow$)

$$(p \hookrightarrow q \wedge [p' \Rightarrow p] \wedge [q \Rightarrow q']) \Rightarrow (p' \hookrightarrow q') \tag{22}$$

In the appendix, we show that the above rules form a proof system for to-always properties which is relatively complete in the sense of Cook.

4 Parallel composition

4.1 Rely

Ideally, properties of the parallel composition of programs should be derivable from the properties of the components. Based on earlier work on rely-guarantee [?] and hypothesis-conclusion in UNITY [?], Chandy [?] proposed the following program property that helps in structured design of programs: Define a function R (for *R*ely) that maps a pair of program properties to a program property, where:

$$R(U,V).F \triangleq (\forall G : U.(F \| G) : V.(F \| G))$$

where U, V are program properties and F and G are programs.

4.1.1 Examples of rely properties

Since $E(p,q).F \Rightarrow E(p,q).F||G$, from (16) we obtain:

$$[E(p,q) \;\Rightarrow\; R\,((p \wedge \neg q) \text{ co } (p \vee q),\; p \leadsto q)]$$

¿From (17):

$$[E(p,q) \Rightarrow R\,((p \wedge \neg q \text{ co } p \vee q \wedge \text{ stable } q),\;\; p \hookrightarrow q)]$$

4.2 Program Component

We shall say that a program F is a component of a program H if there exists a program G such that $H \equiv F||G$. For a program F we introduce a property $component.F$ defined as:

$$component.F.H \triangleq (\exists G : H \equiv F||G) \tag{23}$$

¿From the associativity and idempotence of $||$, we have

$$[component.(F||G) \Rightarrow component.F] \tag{24}$$

$$[component.F||F \equiv component.F] \tag{25}$$

We convert R to a more convenient form, as follows:

$$R\,(U,V).F \;\equiv\; (\forall H : component.F.H \wedge U.H : V.H)$$

and therefore:

$$R\,(U,V).F \;\equiv\; [U \wedge component.F \;\;\Rightarrow\;\; V] \tag{26}$$

and from the predicate calculus:

$$R\,(U,V).F \;\equiv\; [U \Rightarrow (component.F \;\;\Rightarrow\;\; V)] \tag{27}$$

4.3 Weakest Guarantee

Our goal is to explore conjunctive predicate transformers that help in designing parallel programs; so we explore the property transformer wg (for *w*eakest *g*uarantee) where for property V and program F, $wg.F.V$ is a property defined as:

$$[wg.F.V \;\triangleq\;\; \text{weakest } U : R\,(U,V).F]$$

¿From predicate calculus:

$$[\text{ weakest } U : [U \Rightarrow Q] \;=\; Q]$$

and therefore, from (27):

$$[wg.F.V \;\equiv\; component.F \Rightarrow V] \tag{28}$$

Therefore, if property $wg.F.V$ holds for program H, and F is a component of H, then V is a property of H.

¿From (28), $wg.F$ is monotonic, universally conjunctive, universally disjunctive, and idempotent, from which the following formulae regarding property transformer R can be derived. Property transformer R is antimonotonic in its first argument, and monotonic in its second argument. Let U_i and V_i be properties for all $i \in S$, where S is arbitrary.

$$[\,(\forall i \in S : R(U_i, V_i)\,) \;\Rightarrow\; R(\,(\forall i \in S :: U_i),\, (\forall i \in S :: V_i)\,)\,] \tag{29}$$

$$[\,(\forall i \in S :: R(U_i, V_i)\,) \;\Rightarrow\; R(\,(\exists i \in S :: U_i),\, (\exists i \in S :: V_i)\,)\,] \tag{30}$$

Also,

$$[\,R(U, V) \wedge R(V, W) \;\Rightarrow\; R(U, W)\,] \tag{31}$$

$$[\,R(V, V)\,] \tag{32}$$

$$[\,U \Rightarrow V\,] \Rightarrow [\,R(U, V)\,] \tag{33}$$

Further, we have the inheritance theorem [?]:

Inheritance Theorem

$$[wg.F \Rightarrow (\forall G : wg.F \| G)] \tag{34}$$

Proof:

$$
\begin{array}{ll}
 & wg.F.V \\
\equiv & \qquad \{(28)\,\} \\
 & [component.F \Rightarrow V] \\
\Rightarrow & \qquad \{(24)\} \\
 & [component.F \| G \Rightarrow V] \\
\equiv & \qquad \{\,(28)\,\} \\
 & wg.F \| G.V
\end{array}
$$

5 Examples

The detailed proofs for the examples in this section can be found in [?].

5.1 A theorem about progress

In this section, we illustrate reasoning with to-always and Rely properties by proving a useful theorem. First, we state and prove two lemmas.

Lemma 1

$$[R(U,V) \wedge R(U \wedge V, W) \Rightarrow R(U,W)] \tag{35}$$

Proof:

$$\begin{array}{ll} & R(U,V) \\ \Rightarrow & \quad \{ \text{ conjunction (29) with } R(U,U) \text{ (32) } \} \\ & R(U, U \wedge V) \\ \Rightarrow & \quad \{\text{(31) with } R(U \wedge V, W) \ \} \\ & R(U,W) \end{array}$$

Lemma 2

$$\begin{array}{l} [(\forall k : \ x \geq k \wedge x \leq N \hookrightarrow x > k) \wedge (\forall k : \ \text{ stable } x > k)) \\ \Rightarrow \\ (true \hookrightarrow x > N)] \end{array} \tag{36}$$

Proof: By induction on k.

Theorem

$$\begin{array}{l} [\, R((\forall k : \ \text{ stable } x \geq k))\,(\forall k : x \geq k \wedge x \leq N \hookrightarrow x > k) \\ \Rightarrow \\ R((\forall k : \ \text{ stable } x \geq k))\, true \hookrightarrow x > N) \,] \end{array} \tag{37}$$

Proof: Let $U := (\forall k : \ \text{ stable } x \geq k)$, $V := (\forall k : (x \geq k \wedge x \leq N \hookrightarrow x > k))$, and $W := (true \hookrightarrow x > N)$, Then $R(U \wedge V, W)$ follows from (36), and the theorem itself from (35).

5.2 Parallel Sieve of Eratosthenes

In this section, we look at an example, a parallel implementation of a Sieve of Eratosthenes [?]. The program contains a Boolean array a, and on reaching a fixed point[3], and element $a[i]$ should be true if and only if i is prime. Formally,

$$Init \hookrightarrow (\forall i : 0 < i \leq N : prime(i) = a[i]) \tag{38}$$

Init describes a valid initial state which will yield the desired results. It will be derived as part of the proof.

The idea of the solution is that one master process traverses the array. Variable $m : natural$ indicates the next element to examine. On finding an element with $a[i]$, which is potentially prime, an idle process p is started by setting its variable $p.b : natural$ to i. The process then proceeds to falsify all elements of

[3] A program is at a fixed point if its state no longer changes [?]

a which are multiples of i. The next element to falsify is indicated by a variable $p.n$: *natural*. For simplicity, we assume that there are enough processors available.

For convenience, we define the following predicates:

$$[mult(j,k) \equiv (\exists l : 1 < l \leq k : j = l * k)]$$

$$[prime(i) \equiv \neg(\exists m : 1 < m : mult(i,m))]$$

$$[sweep(j,k) \equiv (\forall l : k < m < j : mult(l,k) \Rightarrow \neg a[l])]$$

$$\begin{aligned}&[started(k) \equiv \\ &\quad (\forall l : 1 < l < k : \neg a[l] \vee (\exists p : p.b = l \wedge mult(p.n, p.b) \\ &\quad \wedge sweep(p.n, p.b))]\end{aligned}$$

Informally, $mult(j,k)$ means that j is a multiple of k by some factor other than 1; $prime(i)$ is obvious, and $sweep(j,k)$ means that for all multiples of k which are less than j, the corresponding element of a has been falsified. $started(k)$ means that for all potentially prime elements of a less than k, a process to falsify multiples has been started and is in a valid state. $mult$, $sweep$, and $started$ will be used in the proof to define stable properties of the programs. These stable properties are analogous to loop invariants for sequential programs.

5.2.1 Specification of Slave p

Slave p becomes active when the variable $p.b$ is made nonzero by the master. At the same time, the master initializes the variable $p.n = 2p.b$. The slave then traverses array a, falsifying elements $p.n$ and incrementing $p.n$ by $p.b$. Eventually, all multiples of $p.b$ less than N will be falsified.

Slave p satisfies the following properties:

- A slave takes no actions before it has been started, or after it has terminated.[4]

$$(\forall b : \text{ stable } (p.b = 0) \wedge b)$$

$$(\forall b : \text{ stable } (p.n > N) \wedge b)$$

- A slave does not change the value of m $p.b$, or the b or n component of other slaves.

$$(\forall q,k : \text{ stable } q.b = k) \tag{39}$$

$$(\forall k : \text{ stable } m = k)$$

$$(\forall q,k : q \neq p : \text{ stable } q.n = k)$$

[4] $(p.b = 0 \vee p.n > N)$ is a fixed point of slave p.

- A slave never truthifies any element of a.

$$\text{stable } \neg a[i]$$

- A slave only falsifies the element of a selected by $p.n$

$$p.n \neq i \wedge a[i] \text{ co } a[i] \tag{40}$$

- The condition that $p.n$ is a multiple of $p.b$ will never by falsified, for self or other processes

$$(\forall p : \text{ stable } mult(p.n, p.b)) \tag{41}$$

- All multiples of $p.b$ which are less than $p.n$ are false and $p.n$ is a multiple of $p.b$ is stable.

$$(\forall p : \text{ stable } mult(p.n, p.b) \wedge sweep(p.n, p.b)) \tag{42}$$

- $p.n$ will eventually increase provided that it is non-decreasing in a composed program

$$R\left((\forall k : \text{ stable } p.n > k), (p.n \geq k \wedge p.n \leq N \hookrightarrow p.n > k)\right) \tag{43}$$

These properties are easily proved directly from the text of the one command program given below:

Slave p

if $p.b > 0 \wedge p.n \leq N$ **then** $a[p.n], p.n := false, p.n + p.b$ **end**

The following properties can be derived from those above, and will be used together with the specification of the master to show (38).

- An element of a corresponding to a prime number won't be falsified

$$\text{stable } (\forall p : mult(p.n, p.b) \wedge p.b \neq 1) \wedge prime(i) \wedge a[i]$$

- In a composed program, eventually all multiples of $p.b$ which are at most N will be falsified. The theorem proved in the previous section (37) is used in the proof.

$$\begin{aligned} R\,(&(\forall k : \text{ stable } m \geq k) \\ &\wedge \text{ stable } (mult(p.n, p.b) \wedge sweep(p.n, p.b)), \\ &(mult(p.n, p.b) \wedge sweep(p.n, p.b)) \hookrightarrow sweep(N+1, p.b)). \end{aligned} \tag{44}$$

5.3 Specification of Master

The master satisfies the following properties:

- A master does not change the value of a

  stable $a[i]$

  stable $\neg a[i]$

- The master doesn't "reuse" processors

  $(\forall k : k > 0 : \text{ stable } p.b = k)$

- The master does not falsify the $mult \wedge sweep$ properties of the slaves

  stable $mult(p.n, p.b) \wedge sweep(p.n, p.b)$

- $p.n$ is nondecreasing in master.

  $(\forall k : \text{ stable } p.n \geq k)$

- $p.b$ is never set to 1 by the master

  stable $p.b \neq 1$

- For all $i : i < m$ where $a[i]$ holds, a processor has been dispatched to eliminate multiples of i.

  stable $started(m)$

- m is nondecreasing

  $(\forall k : \text{ stable } m \geq k)$

- m will eventually increase provided it is nondecreasing in a composed program.

  $R\,((\forall k : \text{ stable } m \geq k), (\, m \geq k \wedge m \leq \sqrt{N} \hookrightarrow m > k)\,)$

These properties are satisfied by a master which, at each step, finds a p such that $p.b = 0 \wedge p.n = 0$ and performs the following command:

$$
\begin{array}{lll}
\textbf{if} & a[m] \wedge 1 < m < \sqrt{N} & \textbf{then } p.b, p.n, m := m + 1 \\
\textbf{elsif} & \neg a[m] \wedge 1 < m < \sqrt{N} & \textbf{then } m := m + 1 \\
\textbf{end} & &
\end{array}
$$

¿From these properties, we can conclude

$$R\,((\forall k:\ \text{stable}\ m \geq k) \wedge\ \text{stable}\ started(m),$$
$$started(m) \hookrightarrow started(\sqrt{N}+1)\ )$$

The proof uses (37) and is similar to the proof of (44).

5.4 Properties of composed program

Let *sieve* be the program obtained by composing the master and all of the slaves. From the inheritance theorem for compositional properties (34), and the given specification, the following properties are easily seen to hold for *sieve*:

- $$started(m) \hookrightarrow started(\sqrt{N}+1) \tag{45}$$
- $$(\forall p : (mult(p.n, p.b) \wedge sweep(p.n, p.b)) \hookrightarrow sweep(N+1, p.b)) \tag{46}$$
- $$\begin{array}{l}(\forall p, i : 1 < i \leq N : \\ \text{stable}\ \ (mult(p.n, p.b) \wedge p.b \neq 1) \wedge prime(i) \wedge a[i])\end{array} \tag{47}$$
- $$(\forall k : k > 1 :\ \text{stable}\ p.b = k) \tag{48}$$
- $$\text{stable}\ \neg a[i] \tag{49}$$

One can show that these properties imply the original specification (38) provided that

$$\begin{array}{l} Init \Rightarrow \\ \quad (m = 2 \wedge (\forall p : mult(p.n, p.b) \wedge p.b \neq 1) \wedge \\ \qquad (\forall i : 1 < i \leq N \wedge prime(i) : a[i])) \end{array} \tag{50}$$

The following predicate is easily established by a program during its initialization, and satisfies (50):

$$Init \triangleq (m = 2 \wedge (\forall p : p.b = 0 \wedge p.n = 0) \wedge (\forall i : 1 < i \leq N : a[i])).$$

6 Conclusion

We have defined predicate transformers *wto* and *wg* which can be used to specify progress and compositional properties of parallel programs. The junctivity properties, in particular monotonicity, idempotence, and conjunctivity result in convenient rules for manipulation and calculation with the corresponding properties to-always and Rely. These rules are similar to the rules used with *wp* in sequential programming. We offer *wto* and *wg* as aids to reasoning about parallel programs.

7 Appendix

7.1 Relative completeness of $\hookrightarrow$

In this section, we use show that the rules given so far for proving $\hookrightarrow$ properties are relatively complete in the sense of Cook [?, ?]. In other words, assuming that all valid assertions in the first order logic can be proved, and assuming that our assertion language is expressive enough, we can prove any $\hookrightarrow$ property which holds in the intended model. The proof is similar to the completeness proof for $\leadsto$ given in [?].

¿From [?, ?], we have

$$p \leadsto q \text{ in the intended model} \Rightarrow [p \Rightarrow wlt.q]$$

where

$$[wlt.q \equiv \text{ strongest } r : q \vee we.r],$$

$$[we.r \equiv (\exists s : \text{ s is command } : (we.r)_s)],$$

and

$$[(we.r)_s \equiv \text{ weakest } t : (awp.t \wedge wp.s.r) \vee r]. \tag{51}$$

As a first step, we restrict ourselves to the case where q is stable, so that $[wlt.q \equiv wto.q]$, and construct a proof of $p \hookrightarrow q$. First, we state and prove a lemma.

Lemma

$$\text{stable } q \Rightarrow \text{ stable } we.q \tag{52}$$

Proof:

true

$\Rightarrow$ $\{$ definition of $(we.q)_s\}$

$[(we.q)_s \equiv \text{weakest } t : (awp.t \wedge wp.s.q) \vee q]$

$\equiv$ $\{$ Knaster-Tarski theorem $\}$

$(we.q)_s \equiv \text{weakest } t : [t \Rightarrow (awp.t \wedge wp.s.q) \vee q]$

$\equiv$ $\{$characterization of weakest solution $\}$

$[(we.q)_s \Rightarrow (awp.(we.q)_s \wedge wp.s.q) \vee q] \wedge$
$(\forall p : [p \Rightarrow (awp.p \wedge wp.s.q) \vee q] : [p \Rightarrow (we.q)_s])$

$\Rightarrow$ $\{$ q stable, hence $[q \Rightarrow (awp.q \wedge wp.s.q) \vee q]$,
$q \Rightarrow (we.q)_s, awp$ monotonic, hence $[q \Rightarrow awp.(we.q)_s]$ $\}$

$[(we.q)_s \Rightarrow awp.(we.q)_s \wedge (wp.s.q \vee q)]$

$\Rightarrow$ $\{$ predicate calculus $\}$

$[(we.q)_s \Rightarrow awp.(we.q)_s]$

$\equiv$ { definition of stable }

stable $(we.q)_s$

$\Rightarrow$ { definition of $we.q$, (4) }

stable $we.q$

It is easy to show that $(we.q)_s$ satisfy the hypothesis of (17), thus we can conclude

$$(we.q)_s \hookrightarrow q$$

and from (20)

$$we.q \hookrightarrow q \tag{53}$$

¿From the Knaster-Tarski theorem, and the fact that wto is monotonic, we can write, for stable q:

$$wto.q \equiv (\exists \alpha :: f^{\alpha}.q) \tag{54}$$

where f is defined as

$$[f^0.q \equiv false] \tag{55}$$

$$[f^{i+1}.q \equiv q \vee we.(f^i.q)] \tag{56}$$

$$[f^{\alpha}.q \equiv (\exists \beta : \beta < \alpha : f^{\beta}.q)] \tag{57}$$

Still assuming stable q, we show

$$(\forall \alpha : \text{ stable } f^{\alpha}.q) \tag{58}$$

Proof by transfinite induction:

Base:

stable f^0

$\equiv$ { (55) }

stable $false$

$\equiv$ {definition of stable}

$false \Rightarrow awp.false$

$\equiv$ { predicate calculus }

$true$

Step ordinal

stable $f^{i+1}.q$

$\equiv$ { (56) }

stable $(q \vee we.(f^i.q))$

$\Leftarrow$ { stable q, (4) }

stable $we.(f^i.q)$

$\Leftarrow$ { (52) }

stable $f^i.q$

$\Leftarrow$ {induction hypothesis }

true

Limit ordinal

stable $f^\alpha.q$

$\equiv$ { (57)}

stable $(\exists\beta : \beta < \alpha : f^\beta.q)$

$\Leftarrow$ { stable q, (4) }

$(\forall\beta : \beta < \alpha :$ stable $f^\beta.q)$

$\Leftarrow$ {induction hypothesis }

$true$

Now, we show

$$(\forall\alpha : f^\alpha.q \hookrightarrow q) \tag{59}$$

Proof by transfinite induction:

Base:

$f^0 \hookrightarrow q$

$\equiv$ { (55)}

$false \hookrightarrow q$

$\equiv$ { (17), range of $\exists$ nonempty }

$true$

Step ordinal

$f^{i+1}.q \hookrightarrow q$

$\equiv$ { (56) }

$(q \vee we.(f^i.q)) \hookrightarrow q$

$\Leftarrow$ { (20) }

$(q \hookrightarrow q) \wedge (we.(f^i.q) \hookrightarrow q)$

$\Leftarrow$ { stable q, (19) }

$we.(f^i.q) \hookrightarrow q)$

$\Leftarrow$ { (18) }

$we.(f^i.q) \hookrightarrow f^i.q \wedge f^i.q \hookrightarrow q$

$\Leftarrow$ { stable $f^i.q$, (53) }

$f^i.q \hookrightarrow q$

$\Leftarrow$ {induction hypothesis }

$true$

Limit ordinal

$f^\alpha.q \hookrightarrow q$

$\equiv$ { (57)}

$(\exists\beta : \beta < \alpha : f^\beta.q) \hookrightarrow q$

$\Leftarrow$ { (20) }

$(\forall\beta : \beta < \alpha : f^\beta.q \hookrightarrow q)$

$\Leftarrow$ {induction hypothesis }

$true$

Using this result, we show

$$wto.q \hookrightarrow q$$

Proof:

$$\begin{array}{ll} & true \\ \Rightarrow & \qquad \{(59)\} \\ & (\forall \alpha : f^{\alpha}.q \hookrightarrow q) \\ \Rightarrow & \qquad \{\ (20)\ \} \\ & (\exists \alpha : f^{\alpha}.q) \hookrightarrow q \\ \equiv & \qquad \{(54)\} \\ & wto.q \hookrightarrow q \end{array}$$

We have constructed a proof of $wto.q \hookrightarrow q$. Now be need only prove $[p \Rightarrow wto.q]$ to conclude $p \hookrightarrow q$ and by assumption, all valid assertions can be proved. For arbitrary $q\prime$, we assume that $wst.q\prime$ can be expressed. Since $[wst.q\prime \Rightarrow q\prime]$ is valid in the model, by assumption, it can be proved. Since $wst.q\prime$ is stable, from the above derivation, we can prove $[p \Rightarrow wto.(wst.q\prime)]$ and apply (22) to obtain a proof of $p \hookrightarrow q\prime$. Hence the given proof rules for $\hookrightarrow$ are relatively complete in the sense of Cook.