Conjunctive Predicate Transformers for Reasoning about Concurrent Computation

K. Mani Chandy
Beverly A. Sanders

Computer Science Department
California Institute of Technology

Caltech-CS-TR-93-29
 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 [Dij76]. A computation of a program

*Supported in part by Swiss National Science Foundation grant 5003-034260.
is 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 [CM88]. A program can also be a TLA formula with an initial condition containing only variable type declarations [Lam91]. 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 [Lam91].

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 all wp) defined as:

$$awp.F.q \triangleq (\forall s: s \text{ is a command of } F : wp.s.q)$$ (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.\footnote{Let $s_1 = "x := x + 1", s_2 = "x := x + 2", q = (x = 2), q' = (x = 3). Then, $\forall i awp.q_i = false, awp.q_i = false, awp.q_i \lor q_i = (x = 1).}$

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

$$[awp.F || G.q = awp.F.q \land awp.G.q].$$ (2)
For a predicate \( p \) on program states, we define a program property \( \text{stable} \ p \) as follows.

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

Therefore, \( \text{stable} \ p \) 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:

\[
(\text{stable} \ p).F||G \equiv (\text{stable} \ p).F \land (\text{stable} \ p).G
\]

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

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

\[
(\forall w : w \in W : \text{stable} \ p_w) \Rightarrow \text{stable} \ (\forall w : w \in W : p_w))
\]  
(5)

For any predicate \( q \), we define \( \text{wst}.q \) [San91, Lam87] to be the weakest predicate stronger than \( q \) which is stable in \( F \):

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

Operationally, if \( \text{wst}.F.q \) 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

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

\[
(\text{stable} \ \text{wst}.F.q).F
\]  
(7)

From the Knaster-Tarski theorem [DS90] and the monotonicity of \( awp \)

\[
[\text{wst}.F.q \equiv \text{weakest} \ p : [p \equiv awp.F.p \land q]]
\]

From [DS90], \( \text{wst} \) inherits monotonicity, universal conjunctivity, and or-continuity from \( awp \). Since \( awp \) is universally conjunctive, it is and-continuous and therefore:

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

Thus, for finite state programs, \( \text{wst} \) 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

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

and

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

but

$$\neg((x = 0) \implies (x = 1 \land x = 2)).F$$

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

Knapp [Kna90] proposed a predicate transformer \(\text{wlt} \), for weakest leads to, defined as:

$$p \rightsquigarrow q \text{ in } F \overset{\Delta}{=} [p \Rightarrow \text{wlt}.F.q]$$

where $p$ and $q$ are predicates on states. Knapp showed that \(\text{wlt} \) is monotonic and idempotent but neither finitely conjunctive, finitely disjunctive, and-continuous nor or-continuous.

We define a predicate transformer \(\text{wto}.F \) on predicates of states of $F$ as:

$$\text{wto}.F.q \overset{\Delta}{=} \text{wlt}.F.(\text{wst}.F.q)$$

If \(\text{wto}.F.q \) holds at any point $t$ in an infinite computation of $F$, then there is a point $t'$ in the computation at which \(\text{wst}.F.q \) 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:

$$(p \hookrightarrow q).F \overset{\Delta}{=} [p \Rightarrow \text{wto}.F.q]$$

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 [Cha93].

Since \(\text{wlt} \) and \(\text{wst} \) are monotonic, so is \(\text{wto} \). We shall prove that \(\text{wto} \) is idempotent and finitely conjunctive. We observe that \(\text{wto} \) 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 "\(\text{wlt}.q\)" in place of "\(\text{wlt}.F.q\)" with the understanding that all such statements refer to the same program.

The following three theorems from [Mis92a] and [JKR89] are used in the proofs.
Theorem

\[ q \Rightarrow \text{wlt.q} \] \hspace{1cm} (11)

\[ \neg q \land \text{wlt.q} \Rightarrow \text{awp.wlt.q} \] \hspace{1cm} (12)

\[
\text{stable } r \Rightarrow [(\text{wlt.q} \land r) \Rightarrow \text{wlt.}(q \land r)] \hspace{1cm} (13)
\]

Proof:

\[
\text{stable } q \Rightarrow \text{stable wlt.q} \hspace{1cm} (14)
\]

Corollary

\[ \text{stable wto.q} \] \hspace{1cm} (15)
Theorem (\textit{wto idempotent})

\[ \textit{wto.(wto.q)} \equiv \textit{wto.q} \]

Proof:

\begin{align*}
\textit{wto.(wto.q)} \\
\equiv & \quad \{ \text{definition of wto} \} \\
\textit{wlt.}(\textit{wst.}(\textit{wto.q})) \\
\equiv & \quad \{ \textit{wst.wto.q} = \textit{wto.q}, \text{from (15), (6)} \} \\
\textit{wlt.}(\textit{wto.q}) \\
\equiv & \quad \{ \text{definition of wto, (9)} \} \\
\textit{wlt.(wlt.wst.q)} \\
\equiv & \quad \{ \textit{wlt} \text{ idempotent} \} \\
\textit{wlt.wst.q} \\
\equiv & \quad \{ \text{definition of wto} \} \\
\textit{wto.q} \\
\end{align*}

Theorem (\textit{wto finitely conjunctive})

\[ \textit{wto.q} \land \textit{wto.q'} \equiv \textit{wto.}(q \land q') \]

Proof:

\begin{align*}
\textit{wto.q} \land \textit{wto.q'} \\
\equiv & \quad \{ \text{definition of wto} \} \\
\textit{wlt.wst.q} \land \textit{wto.q'} \\
\Rightarrow & \quad \{ \text{(13),(15)} \} \\
\textit{wlt.}(\textit{wst.q} \land \textit{wto.q'}) \\
\equiv & \quad \{ \text{definition of wto} \} \\
\textit{wlt.}(\textit{wst.q} \land \textit{wlt.wst.q'}) \\
\Rightarrow & \quad \{ \text{(13), stable wst.q (7), with } r := \textit{wst.q}, q := \textit{wst.q}' \} \\
\textit{wlt.wlt.}(\textit{wst.q} \land \textit{wst.q'}) \\
\equiv & \quad \{ \textit{wlt} \text{ idempotent} \} \\
\textit{wlt.}(\textit{wst.q} \land \textit{wst.q'}) \\
\equiv & \quad \{ \textit{wst} \text{ conjunctive} \} \\
\textit{wlt.wst.}(q \land q') \\
\equiv & \quad \{ \text{definition of wto} \} \\
\textit{wto.}(q \land q') \\
\equiv & \quad \{ \text{predicate calculus} \} \\
\textit{wto.}(q \land q') \land \textit{wto.}(q \land q') \\
\Rightarrow & \quad \{ \text{wto monotonic, } (q \land q') \Rightarrow q, (q \land q') \Rightarrow q' \} \\
\textit{wto.q} \land \textit{wto.q'} \\
\end{align*}
3.2 Proofs of \( \rightarrow \) properties

For finite state programs, \( \text{wto} \) 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 \( \text{wto} \) in terms of \( \text{wlt} \) and \( \text{wst} \) allows known results for leads-to to be applied.

From \[\text{Mis92b}\], define the property \( p \co q \),\(^2\) where \( p \) and \( q \) are predicates on program states as:

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

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

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

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

\[
[(p \land \neg q \co p \lor q) \land E(p, q) \Rightarrow p \rightarrow q]
\]  \hspace{1cm} (16)

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

\[
[(p \land \neg q \co p \lor q) \land E(p, q) \land \text{stable } q \Rightarrow p \leftarrow q]
\]  \hspace{1cm} (17)

Additional useful metatheorems are easily obtained from the properties of \( \text{wto} \).

\textbf{Theorem (} \( \leftarrow \text{ is transitive} \))

\[
(p \leftarrow q \land q \leftarrow r) \Rightarrow p \leftarrow r
\]  \hspace{1cm} (18)

\textbf{Proof:}

\[
p \leftarrow q \land q \leftarrow r
\]
\[
\equiv \{\text{definition of } \leftarrow \}\ [p \Rightarrow \text{wto}.q] \land [q \Rightarrow \text{wto}.r]
\]
\[
\Rightarrow \{\text{wto monotonic}\}
\]
\[
[p \Rightarrow \text{wto}(\text{wto}.r)]
\]
\[
\equiv \{\text{wto idempotent}\}
\]
\[
[p \Rightarrow \text{wto}.r]
\]
\[
\equiv \{\text{definition of } \leftarrow \}\ p \leftarrow r
\]

\textsuperscript{2}The familiar \textsc{Unity} property \( p \text{ unless } q \) is equivalent to \( p \land \neg q \co p \lor q \).
Theorem
stable $p \Rightarrow (p \leftrightarrow p)$  

Proof:

stable $p$

$\equiv$

$\{ (6) \}$

$[p \equiv \text{wst}.p]$

$\equiv$

$\{ (11) , q := \text{wst}.p \}$

$[(p \equiv \text{wst}.p) \land (\text{wst}.p \Rightarrow \text{wlt}.\text{wst}.p)]$

$\Rightarrow$

$\{ \text{predicate calculus} \}$

$[p \Rightarrow \text{wlt}.\text{wst}.p]$

$\equiv$

$\{ \text{definition of wto} \}$

$[p \Rightarrow \text{wto}.p]$

$\equiv$

$\{ \text{definition of } \leftarrow \}$

$p \leftarrow p$

Theorem ( $\leftarrow$ universally disjunctive)

For arbitrary set $W$:

$(\forall w : w \in W : p_w \leftarrow q_w) \Rightarrow ((\exists w : w \in W : p_w) \leftarrow (\exists w : w \in W : q_w))$ (20)

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

$[(\forall w : w \in W : p_w \leftarrow q_w) \Rightarrow (\forall w : w \in W : p_w) \leftarrow (\forall w : w \in W : q_w)]$ (21)

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

$(p \leftarrow q \land [pt \Rightarrow p] \land [q \Rightarrow qf]) \Rightarrow (pt \leftarrow qf)$ (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 [Jon86] and hypothesis-conclusion in UNITY [CM88], Chandy [Cha93] proposed the following program property that helps in structured design of programs: Define a function $R$ (for Rely) that maps a pair of program properties to a program property, where:

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

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:

\[
E(p, q) \Rightarrow R((p \land \neg q) \lor (p \lor q), p \sim q)
\]

From (17):

\[
E(p, q) \Rightarrow R((p \land \neg q \lor q \land \text{stable } q, p \sim q))
\]

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 \( \text{component}.F \) defined as:

\[
\text{component}.F.H \triangleq (\exists G : H \equiv F || G)
\]  

From the associativity and idempotence of \( || \), we have

\[
[\text{component}.(F || G) \Rightarrow \text{component}.F]
\]  

\[
[\text{component}.F || F \equiv \text{component}.F]
\]

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

\[
R(U, V).F \equiv (\forall H : \text{component}.F.H \land U.H : V.H)
\]

and therefore:

\[
R(U, V).F \equiv [U \land \text{component}.F \Rightarrow V]
\]

and from the predicate calculus:

\[
R(U, V).F \equiv [U \Rightarrow (\text{component}.F \Rightarrow V)]
\]

4.3 Weakest Guarantee

Our goal is to explore conjunctive predicate transformers that help in designing parallel programs; so we explore the property transformer \( \text{wg} \) (for weakest guarantee) where for property \( V \) and program \( F \), \( \text{wg}.F.V \) is a property defined as:

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

From predicate calculus:

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

and therefore, from (27):
\[ \text{wg}.F.V \equiv \text{component}.F \Rightarrow V \]  
\hspace{100pt} (28)

Therefore, if property \( \text{wg}.F.V \) holds for program \( H \), and \( F \) is a component of \( H \), then \( V \) is a property of \( H \).

From (28), \( \text{wg}.F \) 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.

\[
\left( \forall i \in S : R(U_i, V_i) \right) \Rightarrow R\left( \left( \forall i \in S : U_i \right), \left( \forall i \in S : V_i \right) \right) \]  
\hspace{100pt} (29)

\[
\left( \forall i \in S : R(U_i, V_i) \right) \Rightarrow R\left( \left( \exists i \in S : U_i \right), \left( \exists i \in S : V_i \right) \right) \]  
\hspace{100pt} (30)

Also,

\[
R(U, V) \land R(V, W) \Rightarrow R(U, W) \]  
\hspace{100pt} (31)

\[
R(V, V) \]  
\hspace{100pt} (32)

\[
U \Rightarrow V \Rightarrow R(U, V) \]  
\hspace{100pt} (33)

Further, we have the inheritance theorem [Cha93]:

\section*{Inheritance Theorem}

\[ \text{wg}.F \Rightarrow (\forall G : \text{wg}.F |\!| G) \]  
\hspace{100pt} (34)

Proof:

\[
\text{wg}.F.V
\equiv \{ (28) \}
\]  
\[
\Rightarrow \{ \text{component}.F \Rightarrow V \}
\]  
\[
\Rightarrow \{ \text{component}.F |\!| G \Rightarrow V \}
\]  
\[
\equiv \{ (28) \}
\]  
\[
wg.F |\!| G.V
\]

\section*{5 Examples}

The detailed proofs for the examples in this section can be found in [San93].

\subsection*{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

\[ R(U, V) \land R(U \land V, W) \Rightarrow R(U, W) \]  
(35)

Proof:

\[ R(U, V) \Rightarrow \{ \text{conjunction (29) with } R(U, U) (32) \} \]
\[ R(U, U \land V) \Rightarrow \{ (31) \text{ with } R(U \land V, W) \} \]

Lemma 2

\[ (\forall k : x \geq k \land x \leq N \leftrightarrow x > k) \land (\forall k : \text{stable } x > k) \]
\[ \Rightarrow \]
\[ (\text{true} \leftrightarrow x > N) \]  
(36)

Proof: By induction on \( k \).

Theorem

\[ R((\forall k : \text{stable } x \geq k)) (\forall k : x \geq k \land x \leq N \leftrightarrow x > k) \]
\[ \Rightarrow \]
\[ R((\forall k : \text{stable } x \geq k)) \text{true} \leftrightarrow x > N \]  
(37)

Proof: Let \( U := (\forall k : \text{stable } x \geq k) \), \( V := (\forall k : (x \geq k \land x \leq N \leftrightarrow x > k)) \), and \( W := (\text{true} \leftrightarrow x > N) \), Then \( R(U \land V, W) \) 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 [Bok87]. 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,

\[ Init \leftrightarrow (\forall i : 0 < i \leq N : \text{prime}(i) = a[i]) \]  
(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 \( m : \text{natural} \) 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 \( p.b : \text{natural} \) 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 [CM88]
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:

\[
[\text{mult}(j, k) \equiv (\exists l : 1 < l \leq k : j = l \times k)]
\]

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

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

\[
[\text{started}(k) \equiv
(\forall l : 1 < l < k : \neg a[l]) \lor (\exists p : p.b = l \land \text{mult}(p.n, p.b) \land \text{sweep}(p.n, p.b))]
\]

Informally, \( \text{mult}(j, k) \) means that \( j \) is a multiple of \( k \) by some factor other than \( 1 \); \( \text{prime}(i) \) is obvious, and \( \text{sweep}(j, k) \) means that for all multiples of \( k \) which are less than \( j \), the corresponding element of \( a \) has been falsified. \( \text{started}(k) \) 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. \( \text{mult}, \text{sweep}, \) and \( \text{started} \) 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.\(^4\)

\[
(\forall b : \text{stable } (p.b = 0) \land b)
\]

\[
(\forall b : \text{stable } (p.n > N) \land b)
\]

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

\[
(\forall q, k : \text{stable } q.b = k)
\]

\(^4\( p.b = 0 \lor p.n > N \) is a fixed point of slave \( p \).
\[(\forall k : \text{stable } m = k)\]
\[(\forall q, k : q \neq p : \text{stable } q.n = k)\]

- A slave never truthifies any element of \(a\).
  \[\text{stable } \neg a[i]\]
- A slave only falsifies the element of \(a\) selected by \(p.n\)
  \[p.n \neq i \land a[i] \lor a[i]\] \hspace{1cm} (40)
- The condition that \(p.n\) is a multiple of \(p.b\) will never by falsified, for self or other processes
  \[(\forall p : \text{stable } \text{mult}(p.n, p.b))\] \hspace{1cm} (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.
  \[(\forall p : \text{stable } \text{mult}(p.n, p.b) \land \text{sweep}(p.n, p.b))\] \hspace{1cm} (42)
- \(p.n\) will eventually increase provided that it is non-decreasing in a composed program
  
  \[R((\forall k : \text{stable } p.n > k), (p.n \geq k \land p.n \leq N \leftarrow p.n > k))\] \hspace{1cm} (43)

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

**Slave p**

\[
\text{if } p.b > 0 \land p.n \leq N \text{ then } a[p.n], p.n := \text{false}, p.n + p.b \text{ end}
\]

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
  \[\text{stable } (\forall p : \text{mult}(p.n, p.b) \land p.b \neq 1) \land \text{prime}(i) \land a[i]\]
- 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.
  \[R((\forall k : \text{stable } m \geq k) \land \text{stable } (\text{mult}(p.n, p.b) \land \text{sweep}(p.n, p.b)), (\text{mult}(p.n, p.b) \land \text{sweep}(p.n, p.b)) \leftarrow \text{sweep}(N + 1, p.b)).\] \hspace{1cm} (44)
5.3 Specification of Master

The master satisfies the following properties:

- A master does not change the value of \( a \)
  
  \[
  \text{stable } a[i] \\
  \text{stable } \neg a[i]
  \]

- 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
  
  \[
  \text{stable } mult(p.n, p.b) \wedge sweep(p.n, p.b)
  \]

- \( p.n \) is nondecreasing in master.
  
  \[
  (\forall k : \text{stable } p.n \geq k)
  \]

- \( p.b \) is never set to 1 by the master
  
  \[
  \text{stable } p.b \neq 1
  \]

- For all \( i : i < m \) where \( a[i] \) holds, a processor has been dispatched to eliminate multiples of \( i \).
  
  \[
  \text{stable } started(m)
  \]

- \( m \) is nondecreasing
  
  \[
  (\forall k : \text{stable } m \geq k)
  \]

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

  \[
  R((\forall k : \text{stable } m \geq k), (m \geq k \wedge m \leq \sqrt{N} \rightarrow m > k))
  \]

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:

```plaintext
if \( a[m] \wedge 1 < m < \sqrt{N} \) then \( p.b, p.n, m := m + 1 \)
elsif \( \neg a[m] \wedge 1 < m < \sqrt{N} \) then \( m := m + 1 \)
end
```
From these properties, we can conclude

\[ R((\forall k : \text{stable } m \geq k) \land \text{stable } \mathit{started}(m), \]
\[ \mathit{started}(m) \leftarrow \mathit{started}(\sqrt{N} + 1) \]

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

5.4 Properties of composed program

Let \textit{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 \textit{sieve}:

- \( \mathit{started}(m) \leftarrow \mathit{started}(\sqrt{N} + 1) \) (45)
- \( (\forall p : (\mathit{mult}(p.n, p.b) \land \mathit{sweep}(p.n, p.b)) \leftarrow \mathit{sweep}(N + 1, p.b)) \) (46)
- \( (\forall p, i : 1 < i \leq N : \text{stable } (\mathit{mult}(p.n, p.b) \land p.b \neq 1) \land \mathit{prime}(i) \land a[i]) \) (47)
- \( (\forall k : k > 1 : \text{stable } p.b = k) \) (48)
- \( \text{stable } \neg a[i] \) (49)

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

\[ \text{Init} \Rightarrow \]
\[ (m = 2 \land (\forall p : \mathit{mult}(p.n, p.b) \land p.b \neq 1) \land \]
\[ (\forall i : 1 < i \leq N \land \mathit{prime}(i) : a[i])) \]

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

\[ \text{Init} \triangleq (m = 2 \land (\forall p : p.b = 0 \land p.n = 0) \land (\forall i : 1 < i \leq N : a[i])). \]

6 Conclusion

We have defined predicate transformers \textit{uto} and \textit{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 \textit{wp} in sequential programming. We offer \textit{uto} and \textit{wg} as aids to reasoning about parallel programs.
References


7 Appendix

7.1 Relative completeness of $\Leftarrow$

In this section, we use show that the rules given so far for proving $\Leftarrow$ properties are relatively complete in the sense of Cook [Apt81, Coo78]. 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 $\Leftarrow$ property which holds in the intended model. The proof is similar to the completeness proof for $\Rightarrow$ given in [Rao91].

From [JKR89, Rao91], we have

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

where

\[ \text{wlt.}q \equiv \text{strongest } r : q \lor \text{we.}r, \]
\[ \text{we.}r \equiv (\exists s : s \text{ is command } : (\text{we.}r)s); \]

and

\[ ((\text{we.}r)_s \equiv \text{weakest } t : (awp.t \land wp.s.r) \lor r). \quad (51) \]

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

Lemma

\[ \text{stable } q \Rightarrow \text{stable } \text{we.}q \] \quad (52)

Proof:

\[
\begin{align*}
\Rightarrow & \quad \{ \text{definition of } (\text{we.}q) \_s \} \\
& [(\text{we.}q)_s \equiv \text{weakest } t : (awp.t \land wp.s.q) \lor q] \\
& \equiv \quad \{ \text{Knaster-Tarski theorem } \} \\
& (\text{we.}q)_s \equiv \text{weakest } t : [t \Rightarrow (awp.t \land wp.s.q) \lor q] \\
& \equiv \quad \{ \text{characterization of weakest solution } \} \\
& ((\text{we.}q)_s \Rightarrow (awp.(\text{we.}q)_s \land wp.s.q) \lor q) \land \\
& (\forall p : [p \Rightarrow (awp.p \land wp.s.q) \lor q] : [p \Rightarrow (\text{we.}q)_s]) \\
\Rightarrow & \quad \{ q \text{ stable, hence } [q \Rightarrow (awp.q \land wp.s.q) \lor q], \}
\end{align*}
\]
\( q \Rightarrow (we.q), awp \) monotonic, hence \([q \Rightarrow awp.(we.q)]\}

\[ (we.q)_s \Rightarrow awp.(we.q)_s \land (wp.s.q \lor q) \]

\[ \Rightarrow \{ \text{predicate calculus} \} \]

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

\[ \equiv \{ \text{definition of stable} \} \]

stable \((we.q)_s\)

\[ \Rightarrow \{ \text{definition of } we.q, (4) \} \]

stable \(we.q\)

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

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

and from (20)

\[we.q \Rightarrow q\] (53)

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

\[\text{wto.q} \equiv (\exists \alpha : f^\alpha.q)\] (54)

where \(f\) is defined as

\[ [f^0.q \equiv false] \] (55)

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

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

Still assuming stable \(q\), we show

\[(\forall \alpha : \text{stable } f^\alpha.q)\] (58)

Proof by transfinite induction:

Base:

\[ \text{stable } f^0 \]

\[ \equiv \{ (55) \} \]

\[ \text{stable } false\]

\[ \equiv \{ \text{definition of stable} \} \]

\[ false \Rightarrow awp.false \]

\[ \equiv \{ \text{predicate calculus} \} \]

\[ true \]

Step ordinal

\[ \text{stable } f^{i+1}.q \]

\[ \equiv \{ (56) \} \]
stable \((q \lor \text{we.}(f^i.q))\)
\[\equiv\]
{ stable \(q\), (4) }
stable we.\((f^i.q)\)
\[\equiv\]
{ (52) }
stable \(f^i.q\)
\[\equiv\]
{induction hypothesis }
true

\textit{Limit ordinal}

stable \(f^\alpha.q\)
\[\equiv\]
{ (57) }

stable \((\exists \beta : \beta < \alpha : f^\beta.q)\)
\[\equiv\]
{ stable \(q\), (4) }
\((\forall \beta : \beta < \alpha : \text{stable } f^\beta.q)\)
\[\equiv\]
{induction hypothesis }
true

Now, we show

\((\forall \alpha : f^\alpha.q \leftarrow q)\) \hspace{1cm} (59)

\textit{Proof by transfinite induction:}

\textit{Base:}

\(f^0 \leftarrow q\)
\[\equiv\]
{ (55) }
false \(\leftarrow q\)
\[\equiv\]
{ (17), range of \(\exists\) nonempty }
true

\textit{Step ordinal}

\(f^{\alpha+1}.q \leftarrow q\)
\[\equiv\]
{ (56) }
\((q \lor \text{we.}(f^i.q)) \leftarrow q\)
\[\leftarrow\]
{ (20) }
\((q \leftarrow q) \land (\text{we.}(f^i.q) \leftarrow q)\)
\[\leftarrow\]
{ stable \(q\), (19) }
\(\text{we.}(f^i.q) \leftarrow q\)
\[\leftarrow\]
{ (18) }
\(\text{we.}(f^i.q) \leftarrow f^i.q \land f^i.q \leftarrow q\)
\[\leftarrow\]
{ stable \(f^i.q\), (53) }
\(f^i.q \leftarrow q\)
\[\leftarrow\]
{induction hypothesis }
true

\textit{Limit ordinal}

\(f^\alpha.q \leftarrow q\)
\[\equiv\]
{ (57) }

19
\[(\exists \beta : \beta < \alpha : f^\beta \cdot q) \to q\]
\[
\Leftrightarrow
\begin{cases} \quad (20) \\ \end{cases}
\]
\[(\forall \beta : \beta < \alpha : f^\beta \cdot q) \to q\]
\[
\Leftrightarrow \quad \text{induction hypothesis } \]
\[
\text{true}
\]

Using this result, we show
\[
\text{wto}. q \IFF q
\]

Proof:
\[
\text{true}
\]
\[
\Rightarrow \quad \{(59)\}
\]
\[
(\forall \alpha : f^\alpha \cdot q \to q)
\]
\[
\Rightarrow \quad \{(20)\}
\]
\[
(\exists \alpha : f^\alpha \cdot q) \to q
\]
\[
\equiv \quad \{(54)\}
\]
\[
\text{wto}. q \IFF q
\]

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