Chapter 3
Safety Properties

Jayadev Misra[1]
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712
(512) 471-9547
misra@cs.utexas.edu

April 5, 1994

[1]This material is based in part upon work supported by the Texas Advanced Research Program under Grant No. 003658–219 and by the National Science Foundation Award CCR–9111912.

Contents

Chapter 3

Safety Properties

3.1 Introduction

We study *safety properties* in this chapter. Lamport gives the following informal meaning for safety: bad things do not happen. Formal definitions of safety appear in Alpern and Schneider[1] and Manna and Pnueli[18]. Roughly, a safety property constrains the permitted actions—and, therefore, the permitted state changes—of a program. For instance, requiring that an integer x be nondecreasing in a program prevents any action of the program from decreasing x. Clearly, an action that causes no state change—a *skip*—implements any safety property. We will be particularly interested in several special kinds of safety properties, such as *invariant*, that a predicate remains true at all times during a program's execution, *stable predicate*, that a predicate remains true once it becomes true, and *constant*, that an expression never changes value.

The primary operator for expressing safety properties is **co** (for *co*nstrains). This operator is designed to facilitate reasoning by induction on the number of computation steps. Most safety properties arising in practice are succinctly expressed using **co**. Additionally, **co** has simple manipulation rules which permit easy deduction of new properties from the given ones.

We had adopted *unless*[3] as the primary operator for expressing safety properties, for many years. Our experience (see, particularly, Staskauskas[25]) suggests that the simplicity of formal manipulations is at least as important as the expressive power of an operator. Theoretically, *unless* and **co** are equally expressive, while the latter has more pleasing derived rules that allow simpler manipulations.

Chapter Outline

The operator **co** for a program is defined in terms of the actions of the program. Several special cases of **co**–such as invariant, stable, constant–arise frequently in practice; they are described in Section 3.3. The derived rules for **co**–the main ones being universal conjunctivity and universal disjunctivity–are given in Section 3.4. This section also describes the Substitution Axiom and the Elimination Theorem, which are essential devices for constructions of succinct proofs.

The major part of the chapter, Section 3.5, is devoted to applying the theory. Safety properties for a number of small systems are formalized and manipulated. The examples are chosen from diverse application areas, in order to demonstrate the usefulness of the proposed operator. Section 3.6 contains a few theoretical results. A synopsis of the chapter is in Section 3.7.

3.2 The meaning of co

The systems we consider are (discrete) transition (action) systems. Such a system consists of a number of actions, each of which may, possibly, change the state of the system. A particularly simple view of action systems is given in UNITY[3]. The results of this chapter apply to *any* action system with an arbitrary—finite or infinite—number of actions. The only restriction that we impose is that there be a skip action in the system that may be applied in any program state; the *skip* action does not change the program state.

The safety properties of a system are stated using the *constrains* (**co**) operator. We write p **co** q to denote that whenever p holds, q holds following the execution of the next action.

Given p **co** q it follows that $p \Rightarrow q$, because the action *skip* can be applied in any state satisfying p (and the resulting state, which is same as the previous state, satisfies q). Also, it follows that once p holds q continues to hold upto (and including) the point where p ceases to hold (if ever p ceases to hold). Also, if beyond a certain point p remains *true* forever, so does q (because if p holds then so does q). In any case, once p holds it continues to hold until $\neg p \wedge q$ holds.

We define

$$p \textbf{ co } q \triangleq \langle \forall\, t \;::\; \{p\}\ t\ \{q\} \rangle$$

where t is a transition of the sytem (and the quantification is over all transitions). Here, $\{p\}\ t\ \{q\}$ stands for, if transition t is started in any

system state satisfying p then q holds in the system state upon completion of t, assuming that t completes.

An equivalent formulation of **co** using Dijkstra's wp-calculus[6] is

$$p \ \mathbf{co} \ q \ \stackrel{\Delta}{=} \ \langle \forall \ t \quad :: \quad p \ \Rightarrow \ wlp.t.q \rangle$$

This formulation allows us to establish the derived rules for **co**, in Section 3.3, by exploiting the properties of wlp.

Note on the binding powers of operators The operator **co** has lower binding power than all arithmetic and logical operators. thus,

$$p \ \wedge \ q \ \mathbf{co} \ r \ \wedge \ s$$

is to be interpreted as

$$(p \ \wedge \ q) \ \mathbf{co} \ (r \ \wedge \ s) \qquad \Box$$

Mathematical modeling often consists of converting imprecise or ambiguous informal descriptions to formal descriptions. Such is the case with safety properties which are informally stated using, "... may remain *true as long as* ...," "... can be changed *only if* ...," or "... it is *never the case* that ...." The formal descriptions using **co** are usually easy to construct. Experience helps; therefore, this chapter includes a large number of examples and exercises of varying difficulties. Below, we show some typical informal descriptions and their formal counterparts.

Examples In the following examples, x, y are of type integer.

1. Once x is zero it remains zero until it becomes positive. Other ways of stating such a fact are:

   x can become nonzero only by becoming positive or,
   x can not be decreased if it is zero.

   We observe that for any transition if $x = 0$ is a precondition then either x remains zero or x becomes positive following the transition, i.e.,

   $$x = 0 \ \ \mathbf{co} \ \ x \geq 0$$

2. x remains positive as long as y is. This statement is ambiguous; each of the following formal descriptions represents a reasonable interpretation.

$$\begin{array}{lll} x > 0 \wedge y > 0 & \textbf{co} \; y > 0 & \Rightarrow \; x > 0 \\ x > 0 & \textbf{co} \; y > 0 & \Rightarrow \; x > 0 \\ x > 0 \wedge y > 0 & \textbf{co} \; x > 0 & \end{array}$$

   The first property says that if both x, y are positive they will remain so if y remains positive (we can't tell anything about x if y turns negative). The second property says that if x is positive it will remain so until y is nonpositive. The third property says that if both x, y are positive then the first one to turn nonpositive—if it ever does so—is y.

3. x never decreases. One way to formalize this is to start with the equivalent: if x has a certain value, m, it continues to have that value until it exceeds m. This is identical to Example (1), except that 0 is now replaced by m. That is, for any m

$$x = m \; \textbf{co} \; x \geq m$$

   Here, m is a free variable; the above property is universally quantified over all integers m. Therefore, this property actually represents a set of properties, one property corresponding to each value of m. Another way to express that x never decreases is by,

$$x \geq m \; \textbf{co} \; x \geq m$$

   The formal correspondence between the two properties shown here is the subject of Exercise (7a). The second property is in a particularly important form that will be studied in the next section.

4. x may only be incremented (i.e., increased by 1). This means that in any step, either x retains its old value, say m, or acquires the new value, $m + 1$. With free variable m,

$$x = m \; \textbf{co} \; x = m \; \vee \; x = m + 1$$

5. A line in a delay insensitive circuit is held at its current value until it has been read. Let x be the variable denoting the line value and let y be the variable into which x's value is read. With free variable m

$$x = m \ \wedge \ y \neq m \ \mathbf{co} \ x = m \ \vee \ y = m$$

How does this differ from
$x = m \ \mathbf{co} \ x = m \ \vee \ y = m$
or, $x = m \ \wedge \ y \neq m \ \mathbf{co} \ x = m$?

6. A philosopher may transit from thinking to hungry state. Using predicates t and h to represent that a (particular) philosopher is in thinking or hungry state, respectively, the above says that whenever t is falsified h is established. However,

$$t \ \mathbf{co} \ h$$

is incorrect, because t does not imply h. In such cases, we put the predicate in the lhs as a disjunct in the rhs to form

$$t \ \mathbf{co} \ t \ \vee \ h$$

The above says that whenever t is falsified $(t \ \vee \ h)$ holds; therefore, $\neg t \ \wedge \ (t \ \vee \ h)$ holds in such a state, which implies that h holds whenever t is falsified.

7. A message is received only if it has been sent. Such a statement is best treated by considering its contrapositive: Any message that is unsent remains unreceived. (Hint: Apply contrapositive if you see "only," "only if" or "provided that" in a description.) There are two possible interpretations—using *sent*, *received* to denote that a specific message has been sent, received, respectively—

$$\neg sent \ \wedge \ \neg received \ \mathbf{co} \ \neg received$$
$$\neg sent \ \wedge \ \neg received \ \mathbf{co} \ received \ \Rightarrow \ sent$$

The second property allows for simultaneous truthification of *received* and *sent*—modeling a rendezvous-style communication—whereas the first property prohibits simultaneous transmission and delivery.

8. x, y never change simultaneously. This is an important property that holds in every asynchronous system where x, y are variables that can only be changed by different processes. Using free variables m, n

$$x, y = m, n \ \mathbf{co} \ x = m \ \vee \ y = n$$

That is, at least one of the variables is unchanged by every transition.

9. An integer is added to a set S of integers provided that it exceeds all elements of S. Taking the contrapositive, any integer outside the set that does not exceed all elements stays outside the set. Using free variable m ranging over integers,

$$m \notin S \ \wedge \ m \leq \max.S \ \mathbf{co} \ m \notin S$$

Here $\max.S$ is the value of the largest element of S; it is $-\infty$ if S is empty.

10. Sequence q is changed only by appending an item to its back or removing its front element. This states a fact about how queues are manipulated. Let Q be a free variable, ranging over all sequences of the type of q, and let u be a free variable of item type, (i.e., the type of item that can be appended to q).

$$q = Q \ \mathbf{co} \ q = Q \ \vee \ \langle \exists \ u \ :: \ q = Q; u \rangle \ \vee \ (\exists \ u \ :: \ Q = u; q)$$

Note that simultaneous removal and appending operations are prohibited by this property. □

Relationship of co to other kinds of safety properties One aspect of a safety property is that it can be "implemented" by doing nothing. Therefore, every safety property holds in a system in which *skip* (i.e., do nothing) is the only transition. For such a system, $p \ \mathbf{co} \ q$ holds whenever $p \ \Rightarrow \ q$, because

$$\begin{array}{ll} & p \\ \Rightarrow & \{p \ \Rightarrow \ q \ , \ q = wlp.skip.q\} \\ & wlp.skip.q \end{array}$$

Several standard types of safety properties are simply expressed using **co**. We show some below; others appear in examples and exercises in this chapter.

In [3], we had introduced p *unless* q, for arbitrary predicates p, q, to mean that once p holds it continues to hold as long as q does not; if p and q hold simultaneously nothing can be asserted about the next state. Formally, p *unless* q is, for all transitions t,

$$\{p \ \wedge \ \neg q\} \ \ t \ \ \{p \ \vee \ q\}$$

Since $p \ \wedge \ \neg q \ \Rightarrow \ p \ \vee \ q$, we have

$$p \ \mathit{unless}\ q \ \equiv \ p \ \wedge \ \neg q \ \mathbf{co}\ p \ \vee \ q.$$

Conversely, $p \ \mathbf{co} \ q \ \equiv \ p \ \mathit{unless}\ \neg p \ \wedge \ q$.

To express that p can be falsified only if q holds as a precondition, we write, for all t

$$\{p \ \wedge \ \neg q\} \ \ t \ \ \{p\}$$

This is

$$p \ \wedge \ \neg q \ \mathbf{co}\ p.$$

To express that once p holds it continues to hold until $(\neg p \ \wedge \ q)$ holds, we have, for all t

$$\{p\} \ \ t \ \ \{p \ \vee \ (\neg p \ \wedge \ q)\}$$

Since the rhs is $p \ \vee \ q$, this is

$$p \ \mathbf{co}\ p \ \vee \ q$$

3.3 Special cases of co

3.3.1 Stable, Invariant, Constant

Several special cases of **co** appear frequently in practice; so, we have special names for these cases.

p stable $\equiv p \ \mathbf{co}\ p$
p invariant $\equiv$ **initially** p and p stable
e constant $\equiv \langle \forall \ k \ :: \ e = k \text{ stable} \rangle$
, where e is an expression and k is a free variable of the same type as e.

From above, p stable means that once p is *true* it remains *true* forever, because $p \ \mathbf{co}\ p$ is

$$\langle \forall \ s \ :: \ \{p\} \ \ s \ \ \{p\} \rangle$$

which means that no transition falsifies p. Predicates *false* and *true* are stable in any program. Stable predicates are ubiquitous, for example: The system is deadlocked, a path exists between a sender and a receiver and

the number of messages sent along a channel exceeds a certain value. We will see many instances of stable predicates in Section 3.5.

A predicate is *invariant* for a system if it holds initially and it remains *true* thereafter. Therefore, an invariant predicate is always *true* during a program execution. Predicate *true* is an invariant except in the pathological case where *false* holds initially. The notion of invariant is one of the basic notions in program design. Note that we associate an invariant with a program, not with any point in the program text.

There is a subtle difference between a predicate being invariant and a predicate being "always-true." The difference is analogous to the distinction between "provability" and "truth" in logic. The following example is instructive.

Program *distinction*
  initially $x, y = 0, 0$
  assign $x, y := 0, x$
end

Now, $x = 0 \ \wedge \ y = 0$ is an invariant because it holds initially and

$$\{x = 0 \ \wedge \ y = 0\} \ \ x, y := 0, x \ \ \{x = 0 \ \wedge \ y = 0\}.$$

Since $x = 0 \ \wedge \ y = 0 \ \Rightarrow \ y = 0$, we can assert that $y = 0$ is always *true* during the program execution. However, $y = 0$ cannot be shown to be an invariant, because $y = 0$ cannot be shown stable according to our definition:

$$\{y = 0\} \ \ x, y := 0, x \ \ \{y = 0\}$$

does not hold. In Section 3.4.3, we show that a predicate that is "always-true" is also an invariant, using an extended notion of invariant that employs the "substitution axiom."

An expression, e, is constant if its value never changes during a program execution. Our definition says that once e has a value k, it will continue to have that same value. The reader should verify that the familiar constants—3, *true*, 'HELLO'—are constants according to our definition (see Exercise (1g)). Similar to the above discussion for invariants, there are expressions whose values never change, but that cannot be proven constant according to our definition; for example, y cannot be shown constant in the above program though its value never changes from zero. We can use the substitution axiom, described in Section 3.4.3, to prove that such an expression is constant.

3.3.2 Fixed Point

For a given program, the fixed point predicate, FP, holds upon "termination"; that is, for a state in which FP holds, further execution of the program will not change state, and in a state in which FP does not hold, there is an execution of the program that causes it to change state.

The notion of program termination can be couched in terms of FP: a program is guaranteed to terminate iff it eventually reaches a state that satisfies FP; this is a progress property that is discussed in Chapter 4. The well-known phrase, "program is deadlock-free," means that $\neg FP$ is always *true*; then, it is always possible to change the program state.

We give the following formal characterization of FP in Section 3.6.3: It is the weakest predicate p such that $p \wedge b$ is stable for any b. We will show that there is a unique weakest predicate of this form. The following result is a direct consequence of this definition.

(Stability at Fixed Point) For any predicate b, $FP \wedge b$ stable. □

It is possible to compute FP syntactically from a program text. For a UNITY program, it is obtained by converting each assignment statement to an equality and conjoining all the equalities so obtained.

Examples

Only the transitions in the **assign**-section of each UNITY program is shown.

1. $k := k + 1$

   $FP \equiv k = k + 1$

   $\equiv$ *false*

2. $k := k + 1$ if $k < N$

   $FP \equiv k < N \Rightarrow k = k + 1$

   $\equiv k \geq N$

3. $\langle [\!] \; i : 0 \leq i < N : m := \max(m, A[i]) \rangle$

   $FP \equiv \langle \wedge \; i : 0 \leq i < N : m = \max(m, A[i]) \rangle$

   $\equiv \langle \wedge \; i : 0 \leq i < N : m \geq A[i] \rangle$

   $\equiv m \geq \langle \max \; i : 0 \leq i < N : A[i] \rangle$

3.4 Derived Rules

3.4.1 Basic Rules

All of these rules follow directly from the facts about the predicate transformer, wlp. Here, p, q, p', q', r are arbitrary predicates.

- false **co** p
- p **co** true
- (conjunction; disjunction)

$$\frac{p \textbf{ co } q \ , \ p' \textbf{ co } q'}{\begin{array}{l} p \ \wedge \ p' \textbf{ co } q \ \wedge \ q' \\ p \ \vee \ p' \textbf{ co } q \ \vee \ q' \end{array}}$$

The conjunction and disjunction rules follow from the conjunctivity and monotonicity properties of wlp[6] and of logical implication. These rules generalize in the obvious manner to any set—finite or infinite—of **co**-properties, because wlp and logical implication are universally conjunctive and universally disjunctive. As corollaries of conjunction and disjunction—conjoining r **co** true and disjoining false **co** r, respectively, to p **co** q—we obtain

- (lhs strengthening)

$$\frac{p \textbf{ co } q}{p \ \wedge \ r \textbf{ co } q}$$

- (rhs weakening)

$$\frac{p \textbf{ co } q}{p \textbf{ co } q \ \vee \ r}$$

Also, **co** is transitive

$$\frac{p \textbf{ co } q \ , \ q \textbf{ co } r}{p \textbf{ co } r}$$

This is because $q \ \Rightarrow \ r$ from q **co** r; then rhs of p **co** q can be weakened to p **co** r. Transitivity, however, seems to be of little value because any application of transitivity could be replaced by lhs strengthening—strengthen q to p—or rhs weakening—weaken q to r.

The operator **co** is a form of temporal implication. It shares many of the properties of logical implication, such as the ones shown above. However, it is not reflexive (i.e., p **co** p does not always hold) nor are we allowed to deduce a contrapositive ($\neg q$ **co** $\neg p$ cannot be deduced from p **co** q).

Proof Format for co Since the lhs of a **co**-property can be strengthened and its rhs can be weakened, we can write a deduction in the following format to establish a **co**-property, say p **co** t.

$$
\begin{array}{ll}
 & p \\
\Rightarrow & \{\text{justification}\} \\
 & q \\
\Rightarrow & \vdots \\
 & r \\
\textbf{co} & \{\text{justification}\} \\
 & s \\
\Rightarrow & \vdots \\
\Rightarrow & \{\text{justification}\} \\
 & t
\end{array}
$$

This is not the only proof-format that we employ. In many proofs we deal with several **co**-properties over which conjunctions and disjunctions are employed; in such cases, we write one property per line and associate justifications with each line.

3.4.2 Rules for the special cases

The following rules follow from the conjunction and disjunction rules given above.

(stable conjunction, stable disjunction)

$$
\frac{p \ \textbf{co} \ q \ , \ r \ \text{ stable}}{\begin{array}{l} p \ \wedge \ r \ \textbf{co} \ q \ \wedge \ r \\ p \ \vee \ r \ \textbf{co} \ q \ \vee \ r \end{array}}
$$

A special case of the above is,

$$
\frac{p \ \text{ stable} \ , \ q \ \text{ stable}}{\begin{array}{l} p \ \wedge \ q \ \text{ stable} \\ p \ \vee \ q \ \text{ stable} \end{array}}
$$

Similarly,

$$
\frac{p \ \text{ invariant} \ , \ q \ \text{ invariant}}{p \ \wedge \ q \ \text{ invariant}}
$$

(constant formation)

Any expression built out of constants and free variables is a constant.

3.4.3 Substitution Axiom

An invariant may be replaced by true*, and vice versa, in any property of a program.*

Substitution axiom allows us to deduce properties that we cannot deduce directly from the definition. For instance, given that p **co** q and that J invariant, we can conclude

$p \wedge J$ **co** q , p **co** $q \wedge J$, $p \wedge J$ **co** $q \wedge J$, $p \vee \neg J$ **co** $q \wedge J$, etc.

In particular, given that p is invariant and $p \Rightarrow q$, we can show q invariant, as follows.

p invariant	, given
$p \wedge q$ invariant	, $p = p \wedge q$, since $p \Rightarrow q$
q invariant	, replace p by *true* using the substitution axiom

In Program *distinction* of Section 3.3.1, we showed a predicate that is "always-true" though we could not show it to be invariant. In particular, we could show $x = 0 \wedge y = 0$ invariant, though $y = 0$ could not be proven invariant. Using the argument given above (since $x = 0 \wedge y = 0 \Rightarrow y = 0$), we can now claim, $y = 0$ invariant. Thus, the substitution axiom allows us to remove the distinction between "always-true" and invariant. Exercise 4 asks you to show that "y constant" holds in the program of Section 3.3.1.

Another consequence of the substitution axiom is that a theorem and an invariant have the same status; an invariant can be treated as a theorem, and a theorem, of course, is an invariant. Therefore, we often write simply J, rather than "J invariant."

Note: For compositions of progams, a generalization of the substitution axiom is given in Chapter 6. □

Rationale for the substitution axiom The following rationale for the substitution axiom is due to Knapp[13]. In the definition of p **co** q, we interpreted $\{p\}\ t\ \{q\}$ to mean that if transition t is started in any state satisfying p then q holds in the state upon completion of t, assuming t completes. What is "any state"? We restrict our state space to the *reachable* states: A *reachable state* is a state (i.e., an assignment of values to variables) that may arise during a computation; i.e., either it satisfies the initial condition, or it is a state that may result by applying a transition to a reachable state. It follows that every reachable state satisfies every invariant. In fact, the set of reachable states is the set of states that satisfy

the strongest invariant (we show the existence of the strongest invariant in Section 3.6.2).

We interpret $\{p\}\ t\ \{q\}$, for a given program that includes transition t, to mean that if transition t is started in any reachable state satisfying p then the resulting (reachable) state satisfies q. We may establish $\{p\}\ t\ \{q\}$ by proving, for any invariant J,

$$p\ \wedge\ J\ \Rightarrow\ wlp.t.q$$

This is because the set of states satisfying $p \wedge J$ *includes* all reachable states satisfying p (since a reachable state satisfies any invariant). In particular, we can use the types of variables, or even *true*, as the invariant, J. Even though the notion of invariant is defined using **co**, and the definition of **co** seems to require the notion of invariant, there is no circularity if we start with a known invariant, such as *true* or the variable-types. We could then deduce other invariants which could be applied in deducing other **co**-properties and invariants.

3.4.4 Elimination Theorem

Free variables are essential to our theory. Free variables can be introduced in the lhs by strengthening and in the rhs by weakening, e.g., from p **co** q we can deduce, for program variable x and free variable m,

$p\ \wedge\ x = m$ **co** q and,
p **co** $q\ \vee\ x \neq m$

Free variables can be eliminated by taking conjunctions or disjunctions. We give a useful theorem below for eliminations of free variables by employing disjunction.

Let p be any predicate, x be a program variable and m be a free variable (of the same type as x). Denote by $p[x\ :=\ m]$ the predicate obtained by replacing all occurrences of x by m in p. Now, if p names no program variable other than x then $p[x\ :=\ m]$ has no program variable, and, hence, it is a constant. In particular, $p[x\ :=\ m]$ is stable. Observe that,

$$p = \langle \exists\ m\ :\ p[x\ :=\ m]\ :\ x = m \rangle$$

Theorem(Elimination Theorem)

$$\frac{x = m\ \textbf{co}\ q\ ,\ \text{where } m \text{ is free}\qquad p \text{ does not name } m \text{ nor any program variable other than } x}{p\ \textbf{co}\ \langle \exists\ m\ ::\ p[x\ :=\ m]\ \wedge\ q \rangle}$$

Proof:

$x = m$ **co** q , premise
$p[x := m] \wedge x = m$ **co** $p[x := m] \wedge q$, stable conjunction with $p[x := m]$
$\langle \exists\ m :: p[x := m] \wedge x = m \rangle$ **co** $\langle \exists\ m :: p[x := m] \wedge q \rangle$, disjunction over all m
p **co** $\langle \exists\ m :: p[x := m] \wedge q \rangle$, simplifying the lhs □

The elimination theorem can be applied where x is a list of variables (and m is a list of free variables). In the following example, we apply the theorem with $x = (u, v)$.

Example: Suppose

$$u, v = m, n \textbf{ co } u, v = m, n \ \vee \ (m > n \ \wedge \ u, v = m - 1, n)$$

We will show that $u \geq v$ stable. Using $p \equiv u \geq v$ in the elimination theorem we have

$u \geq v$
co {elimination theorem}
$\langle \exists\ m, n :: (m \geq n \ \wedge \ u, v = m, n) \vee$
$(m \geq n \ \wedge \ m > n \ \wedge \ u, v = m - 1, n) \rangle$
$\Rightarrow$ {simplifying}
$\langle \exists\ m, n :: u \geq v \ \vee \ u \geq v \rangle$
$\Rightarrow$ {simplifying}
$u \geq v$ □

Note on properties and predicates Given that p, q are predicates, p **co** q is a *property*. Though it is tempting to view a property also as a predicate, we will not do so. Therefore, it is syntactically incorrect to write, for instance,

$$p \ \vee \ (q \textbf{ co } r)$$

for predicates p, q, r. We will, however, use the logical symbols, $\wedge$, $\Rightarrow$, $\equiv$ in combining properties. They should be interpreted as follows.

For properties α, β, $\alpha \wedge \beta$ holds in a program provided that both α and β are properties of the program. And, $\alpha \equiv \beta$ holds whenever $\alpha \Rightarrow \beta$ and $\beta \Rightarrow \alpha$ hold. The interpretation of $\alpha \Rightarrow \beta$ is, taking α as an additional premise, β can be deduced from the given program. Equivalently, if Π is the set of *all* properties of the program then β can be deduced from $\Pi \cup \{\alpha\}$.

As a small example, consider the following program over integer variables x, y.

$$x, y \ := \ x+1, y+1 \ [\!] \ x, y \ := \ x-1, y-1$$

It is possible to show for this program that x ascending $\Rightarrow$ y ascending, or, more formally,

$$\langle \forall \ m \quad :: \quad x \geq m \text{ stable} \rangle \ \Rightarrow \ \langle \forall \ n \quad :: \quad y \geq n \text{ stable} \rangle .$$

Similarly,

$$\langle \forall \ m \quad :: \quad x \leq m \text{ stable} \rangle \ \Rightarrow \ \langle \forall \ n \quad :: \quad y \leq n \text{ stable} \rangle .$$

We sketch a proof of the above. For arbitrary integers n, r

$x - y = r$ stable	, from the program text
$x \leq n + r$ stable	, from $\langle \forall \ m \quad :: \quad x \leq m \text{ stable} \rangle$
$x - y = r \ \wedge \ y \leq n$ stable	, conjunction of the above two and simplification
$y \leq n$ stable	, disjunction of above over all r $\Box$

3.5 Applications

We apply our theory to several small problems. In each case, we convert an informal description to a set of **co**-properties, apply some of the manipulation rules given in the preceding section and interpret the derived results.

These exercises suggest that the proposed theory is preferable to intuitive reasoning, not merely for avoiding errors, but also for its simplicity and conciseness.

3.5.1 Nonoperational Descriptions of Algorithms

It is often preferable to describe an algorithm not by a program text but by its properties. There are several advantages to the latter approach: (1) we can express a family of algorithms by one set of properties, because many implementation details can be ignored while writing the properties, (2) it is usually easier to prove facts about an algorithm starting with its properties than from its code, and (3) it is often easier to understand an algorithm from its properties than from its code. We choose a very simple problem from sequential programming—computing the maximum of a set of numbers—to

illustrate these aspects. Section 3.5.2 contains a small exercise of this nature and Section 3.5.3 has a small concurrent programming example using this approach.

The typical algorithm to compute the maximum value, v, of a nonempty finite set, S, of integers is to (1) intially, assign a very small value, $-\infty$, say, to v and (2) then scan the elements of S in some fixed order updating v whenever the scanned element has a larger value. Instead of expressing this algorithm in the notation of a programming language, we describe it by its properties, by focusing on the allowable changes to v.

Using a free variable m that ranges over integers and $-\infty$,

initially $v = -\infty$
$v = m$ **co** $v = m \ \vee \ (v \ \in \ S \ \wedge \ v > m)$

The initial condition is as described above. The **co**-property says that v is only changed to a higher value that is also in S. This description ignores the order in which the elements of S are scanned, leaving open a number of possibilities for implementation (one of which we discuss later in this subsection). Now, we can deduce several properties.

- v is nondecreasing, i.e., for any n

$$v \geq n \quad \text{stable}$$

- v never exceeds the maximum of S

$$v \leq M \text{ invariant} \qquad \text{, where } M = \langle \max\, x \ : \ x \ \in \ S \ : \ x \rangle$$

- $v \ \in \ S$ stable

The proofs of all these properties appeal to the elimination theorem; we show one below.

$v \leq M$ invariant:

initially $v \leq M$, from $v = -\infty$

To show that $v \leq M$ stable,

$v \leq M$
co {elimination theorem on: $v = m$ **co** $v = m \ \vee \ (v \ \in \ S \ \wedge \ v > m)$}

$$
\begin{array}{ll}
 & \langle \exists\ m \ ::\ (m \leq M\ \wedge\ v = m)\ \vee\ (m \leq M\ \wedge\ v\ \in\ S\ \wedge\ v > m) \rangle \\
\Rightarrow & \{\text{weakening}\} \\
 & \langle \exists\ m \ ::\ v \leq M\ \vee\ v\ \in\ S \rangle \\
\Rightarrow & \{\text{simplifying}\} \\
 & v \leq M\ \vee\ v\ \in\ S \\
\Rightarrow & \{v\ \in\ S\ \Rightarrow\ v \leq M\} \\
 & v \leq M
\end{array}
$$

Note that we can't yet prove that v will eventually equal M; wait until we develop the theory of progress in Chapter 4.

A refinement of this algorithm is the following. The set S is represented by an array $A[0..]$; the elements of A are scanned in the order $A[0], A[1] \ldots$ updating v appropriately. We again describe this algorithm by its properties, i.e., by the way its variables are manipulated. Let s be the index into A upto (but not including) which the elements have been scanned. Then,

$$
\begin{array}{l}
\textbf{initially}\ v, s = -\infty, 0 \\
v, s = m, k\ \textbf{co}\ v, s = m, k\ \vee\ v, s = \max(m, A[k]), k+1
\end{array}
$$

where m ranges over integers and $-\infty$, and k ranges over the indices of A. We can prove that v is the maximum over the scanned segment, i.e.,

$$v = \langle \max\ i\ :\ 0 \leq i < s\ :\ A[i] \rangle$$

Initially the above holds, since $-\infty = \langle \max\ i\ :\ 0 \leq i < 0\ :\ A[i] \rangle$. We now prove the stability of $v = \langle \max\ i\ :\ 0 \leq i < s\ :\ A[i] \rangle$.

$$
\begin{array}{ll}
 & v = \langle \max\ i\ :\ 0 \leq i < s\ :\ A[i] \rangle \\
\textbf{co} & \{\text{elimination theorem}\} \\
 & \langle \exists\ m, k \quad ::\quad [m = \langle \max\ i\ :\ 0 \leq i < k\ :\ A[i] \rangle\ \wedge\ v, s = m, k] \\
 & \qquad\qquad \vee\quad [m = \langle \max\ i\ :\ 0 \leq i < k\ :\ A[i] \rangle\ \wedge \\
 & \qquad\qquad\qquad\qquad v, s = \max(m, A[k]), k+1] \\
 & \rangle \\
\Rightarrow & \{\text{arithmetic}\} \\
 & v = \langle \max\ i\ :\ 0 \leq i < s\ :\ A[i] \rangle
\end{array}
$$

Note that for this trivial problem the formal proof is trivially simple.

3.5.2 Common Meeting Time

This problem has been discussed in Chandy and Misra[3, Section 1.4]. The purpose of this example, much like the one in Section 3.5.1, is to explore a family of design alternatives by considering the safety properties common to all members of the family.

It is required to find the earliest meeting time acceptable to every member in a group. Time is nonnegative and real-valued (though, in Section 4.5.2 we restrict it to be integer-valued). To simplify notation, assume that there are only two persons, F and G, in the group. Associated with F, G are functions, f, g, respectively, where each function maps nonnegative reals to nonnegative reals (i.e., times to times). For any real t, $f(t)$ is the earliest time at or after t when F can meet; similarly $g(t)$ is defined. Time t is acceptable to F iff $f(t) = t$. A *common meeting time*, t, is acceptable to both F and G, i.e., $f(t) = t \ \wedge \ g(t) = t$. The goal is to design an algorithm that computes the earliest (i.e., smallest nonnegative) common meeting time, provided one exists.

Several algorithms and their implementations on various architectures have been described in [3]. Here, we define the essential safety properties common to *all* these algorithms.

First, we have to make certain assumptions about f, g so that the earliest meeting time can be computed effectively. From the problem description, for all nonnegative real m, n

$$\text{(ascending)} \quad m \leq f(m), \ m \leq g(m) \tag{CMT0}$$

Also, we postulate that f, g are monotonic.

$$\begin{array}{ll} \text{(monotonic)} & m \leq n \ \Rightarrow \ f(m) \leq f(n) \\ & m \leq n \ \Rightarrow \ g(m) \leq g(n) \end{array} \tag{CMT1}$$

We postulate that there is a variable t, nonnegative and real, whose value never exceeds the earliest common meeting time, and variable t is increased, eventually, if it is not a common meeting time. The latter property is discussed in Section 4.5.2. Here, we consider the safety aspect of the problem, given by the first requirement on t. A strategy for implementing this requirement is to set t to 0, initially. The rule for modifying t is: if t's value is m before a transition then it must not exceed both $f(m)$ and $g(m)$ after the transition. It is not obvious that this strategy prevents t from exceeding the earliest common meeting time; our proof of this fact is quite short.

The formal description of the strategy is as follows.

$$\textbf{initially } t = 0 \tag{CMT2}$$

$$t = m \ \textbf{co} \ t \leq \max(f(m), g(m)) \tag{CMT3}$$

Let $com(n)$ denote that n is a common meeting time, i.e., $f(n) = n \ \wedge \ g(n) = n$. We prove, from CMT0-CMT3,

$$com(n) \ \Rightarrow \ t \leq n \tag{CMT4}$$

i.e., any common meeting time–in particular, the earliest common meeting time–is at least t.

The property (CMT4) is an invariant. It holds initially for any nonnegative real n, using (CMT2). The remaining proof obligation is

$$(com(n) \;\Rightarrow\; t \leq n) \text{ stable} \qquad \text{(CMT5)}$$

which we proceed to prove. Applying the elimination theorem on (CMT3)

$$com(n) \;\Rightarrow\; t \leq n \;\mathbf{co}\; \langle \exists\, m \;::\; (com(n) \;\Rightarrow\; m \leq n) \wedge t \leq \max(f(m), g(m)) \rangle$$

We show that the term in the rhs, $(com(n) \;\Rightarrow m \leq n) \wedge t \leq \max(f(m), g(m))$, implies $com(n) \;\Rightarrow\; t \leq n$.

$com(n)$
$\Rightarrow$ {using $com(n) \;\Rightarrow\; (m \leq n)$}
$m \leq n$
$\Rightarrow$ {monotonicity of f, g from (CMT1)}
$f(m) \leq f(n) \;\wedge\; g(m) \leq g(n)$
$\Rightarrow$ {from $com(n) \;:\; f(n) = n \;\wedge\; g(n) = n$}
$f(m) \leq n \;\wedge\; g(m) \leq n$
$\Rightarrow$ {arithmetic}
$\max(f(m), g(m)) \leq n$
$\Rightarrow$ {using $t \leq \max(f(m), g(m))$, the second conjunct in the rhs}
$t \leq n$

Thus

$$com(n) \;\Rightarrow\; t \leq n \;\mathbf{co}\; \langle \exists\, m \;::\; com(n) \;\Rightarrow\; t \leq n \rangle$$
or, $$com(n) \;\Rightarrow\; t \leq n \;\mathbf{co}\; com(n) \;\Rightarrow\; t \leq n$$

establishing (CMT5).

The strategy given by (CMT3) is quite general. It subsumes the following strategies.

$$t = m \;\mathbf{co}\; t = m$$
$$t = m \;\mathbf{co}\; t = m \;\vee\; t = \max(f(m), g(m))$$
$$t = m \;\mathbf{co}\; t = m \;\vee\; t = f(m) \;\vee\; t = g(m)$$

In each of the above properties, the rhs is stronger than that of (CMT3); hence, (CMT3)—and, consequently, (CMT5)—can be derived from each of the above properties. A weak safety property like (CMT3) is preferable for

initial design explorations because it constrains the allowable transitions only minimally.

The property (CMT3) does not guarantee that t is nondecreasing. It is useful to have nondecreasing t if, for instance, we wish to guarantee that any change in t brings it closer to the earliest common meeting time. Therefore, we postulate the following stronger version of (CMT3).

$$t = m \textbf{ co } m \leq t \leq \max(f(m), g(m)) \qquad \text{(CMT6)}$$

This property implies—by weakening its rhs—that

$$t = m \textbf{ co } m \leq t$$

which is equivalent to (see Exercise 7a)

$$t \geq m \text{ stable.}$$

Each of the following programs (in which the initial condition, $t = 0$, is not shown) implements (CMT6).

P0 :: $t := t$ {does nothing useful}
P1 :: $t := f(t)$ [] $t := g(t)$
P2 :: $t := \max(f(t), g(t))$
P3 :: $t := t + 1$ if $t + 1 \leq \max(f(t), g(t))$

Van de Snepscheut[26] showed that Program (P1), above, satisfies (CMT5) using only the assumption of monotonicity of f, g (CMT1). Our derivation suggests that the requirement that f, g be ascending, (CMT0), can be replaced by the weaker $m \leq \max(f(m), g(m))$, for all nonnegative real m.

3.5.3 A Small Concurrent Program: Token Ring

The method advocated in the last section—describing an algorithm by its properties—is most profitably applied to concurrent algorithms. Here, we consider a simple mutual exclusion problem.

A set of processes are connected in a ring where the message transmissions take place over the edges of the ring; thus a process may communicate only with its left or right neighbor. It is given that at most one process may transmit at any time. A technique to meet this requirement is to have a single token circulate in the ring and allow only the token-holder to transmit. The following is an abstraction of this solution.

A process is in one of three states: waiting to transmit (also called *hungry*), transmitting (also called *eating*) and neither of the above (also

called *thinking*). The terms hungry, eating, thinking are taken from the dining philosophers problem which is a standard abstraction in resource allocation. An eating process can transit only to thinking (see $tr1$, below), a thinking process can transit only to hungry (see $tr2$) and a hungry process can transit only to eating ($tr3$); a hungry process remains hungry as long as it does not hold the token ($tr4$); the token can be relinquished only if the process is noneating ($tr5$). We would like to prove that there is at most one eating process—i.e., at most one transmitting process—in the system. An intuitive proof is obvious: There is at most one token in the system and an eating process holds the token. Unfortunately, the proof is far from obvious; for instance, the rule for relinquishing the token, as stated above, is so vague that it allows a hungry process to release the token when it transits to eating. Furthermore, initially the eating process, if any, has to hold the token ($tr0$). Such details, which are often ignored in informal descriptions, cause endless problems.

Notation: We write $h_i/e_i/t_i$ to denote that process i is hungry/eating/thinking. These predicates are mutually exclusive and $h_i \vee t_i \vee e_i \equiv true$. The position of the token is in the variable p, i.e., $p = i$ (as in $tr5$) denotes that process i holds the token. □

In the following, i ranges over all processes.

initially $e_i \Rightarrow p = i$	$(tr0)$
e_i **co** $e_i \vee t_i$	$(tr1)$
t_i **co** $t_i \vee h_i$	$(tr2)$
h_i **co** $h_i \vee e_i$	$(tr3)$
$h_i \wedge p \neq i$ **co** h_i	$(tr4)$
$p = i$ **co** $p = i \vee \neg e_i$	$(tr5)$

The properties, $tr0$–$tr5$, specify only the safety aspects of the system. In particular, the protocol for transmitting the token (which is sent to the neighbor in a specific direction in the ring) is completely ignored in this specification. The complete protocol is specified in Section 4.5.3. Here, we show that the partial specification is sufficient for establishing mutual exclusion, i.e., that there is at most one eating process.

To show that there is at most one eating process, we prove, for all i, j

$$e_i = e_j \Rightarrow i = j$$

The result follows by showing that an eating process holds the token, i.e., for any i, $e_i \Rightarrow p = i$. Then $(e_i \wedge e_j) \Rightarrow (p = i \wedge p = j) \Rightarrow (i = j)$. To show that $e_i \Rightarrow p = i$, for any i, initially the result follows from $tr0$.

The remaining proof obligation is to show that $(e_i \ \Rightarrow \ p = i)$ is stable, for any i.

$t_i \ \vee \ h_i \ \vee \ p = i \ \mathbf{co} \ t_i \ \vee \ h_i \ \vee \ p = i \ \vee \ \neg e_i$
, disjunction of $(tr2, tr4, tr5)$
$e_i \ \Rightarrow \ p = i$ stable , replacing $t_i \ \vee \ h_i$ by $\neg e_i$ and rewriting

Observe that $(tr1, tr3)$ are unnecessary for the proof of mutual exclusion. This proof is concise partly because it does not mimic the usual common-sense reasoning. The proof demonstrates the effectiveness of the disjunction rule.

3.5.4 From Program Texts to Properties

Sometimes portions of a system are specified by program fragments whereas other portions are specified by properties. It may be useful, therefore, to convert a program text to properties. This conversion can be carried out automatically, especially for programs in which each statement corresponds to a transition. We sketch, below, how this is done.

Let x denote a group of program variables. Consider all the statements that can change one or more variables in x. If these statements are of the form

$$x \ := \ e_1 \quad [\!] \ldots \ x \ := \ e_i \ \ldots \ [\!] \ \ldots$$

then the current value of x, say m, can be changed to $e_1[x \ := \ m]$—where $e_1[x \ := \ m]$ is the expression obtained by replacing x by m in e_1—by executing the statement $x \ := \ e_1$, or, in general, to $e_i[x \ := \ m]$ by executing $x \ := \ e_i$. Since x can be changed only by the above statements we get

$$x = m \ \mathbf{co} \ x = m \ \vee \ x = e_1[x \ := \ m] \ \vee \ \ldots \ x = e_i[x \ := \ m] \ \vee \ \ldots$$

Note that e_i may name variables outside x.

A common syntactic variation is a guarded command of the form

$g_i \ \rightarrow \ x \ := \ e_i$
or, $x \ := \ e_i$ if g_i (in UNITY).

Given a set of statements of this form that can modify x, we obtain

$$x = m \ \mathbf{co} \ x = m \ \vee \ (g_1[x \ := \ m] \ \wedge \ x = e_1[x \ := \ m]) \ \vee \ \ldots \\ (g_i[x \ := \ m] \ \wedge \ x = e_i[x \ := \ m]) \ \vee \ \ldots$$

In systems consisting of multiple processes in which a variable is changed by at most one process, the variables are naturally partitioned among the processes, each variable *belonging* to the process that may change it (the variables that are never changed are constants). For each process, we obtain a **co**-property in the above manner over the variables that it may change. For example, given below is a system that represents two processes, one of which may modify x and the other y (though the latter can read x):

$$x := x+1 \;[\!]\; y := y+1 \quad \text{if} \quad x > y$$

We obtain, for the two processes,

$$x = m \;\mathbf{co}\; x = m \;\vee\; x = m+1 \quad \text{and,}$$
$$y = n \;\mathbf{co}\; y = n \;\vee\; (x > n \;\wedge\; y = n+1)$$

In order to prove a safety property for this transition system—say, $x \geq y$ stable—we could either start with the program text (and show that each transition preserves the property) or we could start with the safety property of the system obtained as above and then deduce the result. Whenever we have a mixture of programming fragments and properties, the latter strategy is preferable since we can then work using a single notation.

3.5.5 Finite State Systems

Finite state descriptions are often used for specifications of communication protocols and discrete control systems. Here, we show how the state transitions can be described in our theory. As an example of a finite state system we consider an extremely simplified version of a telephone system; see Staskauskas[25] for a more realistic version.

We focus our attention on a single telephone. We postulate it to have the following states:

idle: the handset is on-hook and not ringing

ring: the handset is on-hook and ringing

call: the handset is off-hook and a number is being dialed

conn: the telephone is engaged in a call

rej: the telephone is receiving a busy signal or the other party has disconnected

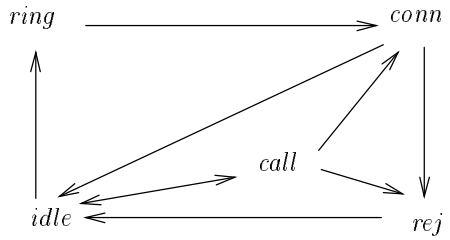


Figure 3.1: A State Transition Diagram for a Telephone

Our model is so simple that we can't even distinguish in the *conn* state if this telephone initiated a call or received a call.

A possible state transition diagram is shown in Figure 3.1. Most of the transitions are self-explanatory; for instance, the transition from *conn* to *idle* is taken when the user hangs up and the transition from *call* to *rej* is effected by the telephone switch giving a busy signal to the caller.

The safety properties of a finite state system can be obtained automatically as follows. There is one safety property for each state. For a state x that has possible transitions to states y and z, the corresponding property is

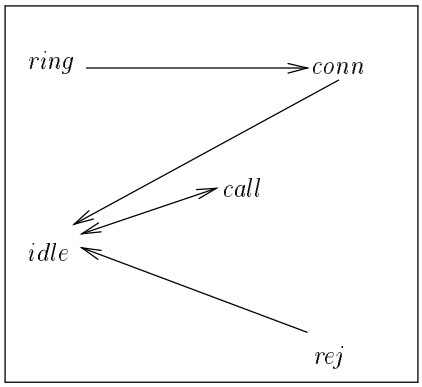
$$\text{state} = x \;\; \textbf{co} \;\; \text{state} = x \;\vee\; \text{state} = y \;\vee\; \text{state} = z$$

which expresses the fact that from the current state, x, a transition can effect a state change to y or z only. The safety properties obtained from the diagram in Figure 3.1 are as follows (in these properties we write, for instance, *idle* to denote state=*idle*).

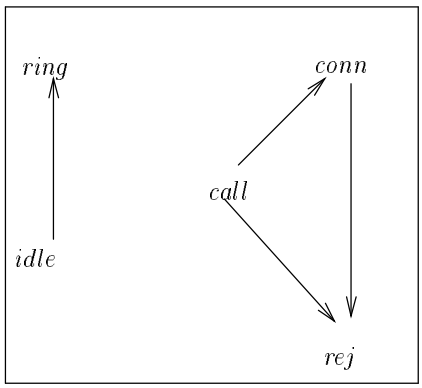
$$\begin{array}{lll} \mathit{idle} & \textbf{co} & \mathit{idle} \vee \mathit{ring} \vee \mathit{call} \\ \mathit{call} & \textbf{co} & \mathit{call} \vee \mathit{idle} \vee \mathit{conn} \vee \mathit{rej} \\ \mathit{conn} & \textbf{co} & \mathit{conn} \vee \mathit{idle} \vee \mathit{rej} \\ \mathit{rej} & \textbf{co} & \mathit{rej} \vee \mathit{idle} \\ \mathit{ring} & \textbf{co} & \mathit{ring} \vee \mathit{conn} \end{array}$$

The transitions in Figure 3.1 are of two types—those made by the user and those made by the telephone switch. For instance, the transition from *idle* to *ring* is made by the telephone switch (the user can never make a telephone ring) whereas all transitions to *idle* are made by the user. Figure 3.2 depicts the transition diagrams for the user and the switch. Clearly, properties can be written down for each of these figures as shown above.

In these diagrams, we have not shown the conditions under which a transition takes place. For instance, the switch causes a transition from *call* to *conn* only if a connection has been made to the dialed number, and



3.2a: Transitions Made by the User



3.2b: Transitions Made by the Switch

Figure 3.2: Partitioning the Transitions of Figure 3.1 between the User and the Switch

the transition from *conn* to *idle* by the user is made as a result of the user hanging up. Sometimes, the condition cannot be expressed as a finite state property. To see this, let *num* be a variable in which the number dialed by the user is stored (in this, extremely trivial, description we ignore the fact that digits are stored one by one into *num*; instead, we assume that the entire number is stored into *num* in a single atomic action). The variable *num* can also take on a special value, $-$, denoting that no number has been stored in *num*.

A precondition for transition of a user u from *call* to *conn* is that

its *num* differs from $-$, and
its called number, v, is in *ring* state

A postcondition of this transition is that u, v are both in *conn* states or that u hangs up. This fact can be expressed by (where we prefix states by the user identity to avoid confusion), for all users u, v $(u \neq -, v \neq -)$

$$u.call \wedge u.num = v \wedge v.ring \ \mathbf{co} \ (u.call \wedge u.num = v \wedge v.ring) \vee u.idle \vee (u.conn \wedge v.conn \wedge u.num = v)$$

This is, however, a very coarse property; we cannot even state that a user is connected to exactly one party in the *conn* state. By suitably introducing other variables (for instance, *u.party* to denote the party to which u is connected in the *conn* state; then, $u.conn \wedge u.num \neq - \Rightarrow u.party = u.num$) we can obtain a more refined specification.

It becomes clear as we add more and more details to a specification that the finite state diagram provides a gross description of the flow of control. It is sometimes appealing to work with a diagram because a visual check may suffice to deduce a property. However, many of these systems have aspects that cannot be captured by a finite number of states.

3.5.6 Auxiliary Variables

In stating and verifying properties of programs, it is sometimes necessary to introduce *auxiliary* variables, variables whose values at any point in the computation depend only on the history of the other variable values. For instance, for u, an integer valued variable, define an auxiliary variable v whose value is the number of times that u's value has changed during the course of a computation. Now, v can be introduced by defining it to be initially zero, and modifying the program text to increment v whenever u is changed. The problem with this approach is that the relationship between u, v as stated above is lost; it has to be gleaned from the program text.

Since we now have a logical operator to relate values of various variables over the course of a computation, we can define v directly, as follows; here, m, n are integer-valued free variables.

initially $v = 0$
$u, v = m, n$ **co** $u, v = m, n \ \vee \ (u \neq m \ \wedge \ v = n + 1)$

This property asserts that every change in u is accompanied by an increment in the value of v and, as long as u does not change v does not change either. It is now a simple matter to prove various facts about v, such as that v is nondecreasing.

A useful auxiliary variable is the sequence of distinct values written into a given variable; for a variable x let $\widehat{x}$ be this sequence. Then, for m, n of the appropriate type

initially $\widehat{x} = \langle\langle x \rangle\rangle$ $\quad \{\langle\langle x \rangle\rangle$ is the sequence consisting of the value of $x\}$
$x, \widehat{x} = m, n$ **co** $x, \widehat{x} = m, n \ \vee \ (x \neq m \ \wedge \ \widehat{x} = n; x)$

where ";" is the sequence concatenation operator.

Finally, we show that for a semaphore s, $s - nv + np$ is constant where np, nv are the numbers of successful "P" and "V" operations, respectively. We define np, nv as follows, where m, n, r are integer-valued free variables.

initially $np, nv = 0, 0$
$s, nv = m, n$ **co** $n \leq nv \leq n + 1 \ \wedge \ (s = m + 1 \ \equiv \ nv = n + 1)$
$s, np = m, r$ **co** $r \leq np \leq r + 1 \ \wedge \ (s = m - 1 \ \equiv \ np = r + 1)$

We know that for semaphore s

$$s = m \textbf{ co } m - 1 \leq s \leq m + 1$$

We know other facts about semaphores, such as $s \geq 0$, that will not be required in the proof. Nor will we use the initial condition. Conjoining the above three **co**-properties

$$s, nv, np = m, n, r \textbf{ co } m - 1 \leq s \leq m + 1 \ \wedge \ n \leq nv \leq n + 1 \wedge \\ r \leq np \leq r + 1 \ \wedge \ (s = m + 1 \ \equiv \ nv = n + 1) \wedge \\ (s = m - 1 \ \equiv \ np = r + 1)$$

Weaken the rhs to $s - nv + np = m - n + r$. Then,

$$s, nv, np = m, n, r \textbf{ co } s - nv + np = m - n + r.$$

Applying the elimination theorem,

$$s - nv + np \quad \text{constant}$$

3.5.7 Deadlock

A trivial form of deadlock—captured by the "after you, after you" paradigm —arises in a cycle of processes when each process waits for its left neighbor. It is intuitively obvious that this system state will persist forever. However, a formal proof or even a rigorous argument based on this verbal description is messy. Most such proofs are by contradiction: If this system state does not persist forever then there is a process that stops waiting; let x be the first process to stop waiting; x can stop waiting only if the process for which it has been waiting is not waiting; therefore, there is a process that stopped waiting earlier than x, contradicting our choice of x as the first process to stop waiting. Such informal arguments, though intuitively-appealing, are neither precise about the assumptions they make—for instance, can x, y, where x is waiting for y, stop waiting simultaneously—nor are they concise.

Process x waits for y means that whenever both x, y are waiting both will continue waiting until y stops waiting. This informal description is quite ambiguous; it admits at least two interpretations as shown below. Denoting by xw that process x is waiting (and similarly, for yw) x waits for y means

$$xw \wedge yw \textbf{ co } xw \qquad \text{(strong wait) or,} \qquad \text{(D1)}$$
$$xw \wedge yw \textbf{ co } yw \Rightarrow xw \qquad \text{(weak wait)} \qquad \text{(D2)}$$

In strong wait, both processes cannot stop waiting simultaneously; in weak wait they could. By suitably weakening the rhs of strong wait we can obtain weak wait. Therefore, strong wait is indeed stronger than weak wait.

Now, it is trivial to show that two processes, x, y, waiting strongly for each other are deadlocked. Because, we have then,

$$xw \wedge yw \textbf{ co } xw$$
$$yw \wedge xw \textbf{ co } yw$$

Applying conjunction we get

$$xw \wedge yw \textbf{ co } xw \wedge yw$$

We conclude from the above that $xw \wedge yw$ is stable; hence $xw \wedge yw$, once *true*, persists forever. Our proof avoids explicit arguments about time, or arguments based on contradiction. The essence of the proof is an induction on the number of computation steps, which is captured in the **co**-properties. Now, we prove the result for any finite cycle of processes, but we will do more. We show that if one process in a cycle strong-waits and the remaining processes weak-wait then a deadlock arises when all processes are waiting.

Let the processes be indexed 0 through N, $0 < N$. Let w_i denote that process i is waiting. Suppose process 0 strong-waits for process N; from (D1) this is expressed by

$$w_0 \ \wedge \ w_N \ \ \mathbf{co} \ \ w_0 \tag{D3}$$

In the remaining processes, process $(i+1)$ weak-waits for process i, $0 \leq i < N$. This is expressed, as in (D2), by

$$\langle \forall \ i \ : \ 0 \leq i < N \ : \ w_{i+1} \ \wedge \ w_i \ \ \mathbf{co} \ \ w_i \ \Rightarrow \ w_{i+1} \rangle \tag{D4}$$

This completes the mathematical modeling of the problem. Now come the formal manipulations. Conjunction of (D4), over all i, $0 \leq i < N$, yields

$$\langle \forall \ i \ : \ 0 \leq i < N \ : \ w_{i+1} \ \wedge \ w_i \rangle \ \ \mathbf{co} \ \ \langle \forall \ i \ : \ 0 \leq i < N \ : \ w_i \ \Rightarrow \ w_{i+1} \rangle$$

Conjunction of the above and (D3) yields, after simplifying the lhs,

$$\langle \forall \ i \ : \ 0 \leq i \leq N \ : \ w_i \rangle \ \ \mathbf{co} \ \ w_0 \ \wedge \ \langle \forall \ i \ : \ 0 \leq i < N \ : \ w_i \ \Rightarrow \ w_{i+1} \rangle .$$

The rhs, using the induction principle, implies $\langle \forall \ i \ : \ 0 \leq i \leq N \ : \ w_i \rangle$. Therefore,

$$\langle \forall \ i \ : \ 0 \leq i \leq N \ : \ w_i \rangle \ \text{ stable}$$

Contrast this argument with a typical informal argument of the following kind. Consider the state of the system where all processes are waiting. If any process $(i+1)$, $i \geq 0$, stops waiting subsequently then process i must have stopped waiting then or before $(i+1)$. Applying the same argument to i, $i-1$, etc., we conclude that process 0 stops waiting at or before any other process stops waiting. However, process N has to stop waiting strictly prior to process 0. Applying the same kind of argument from process N down to $(i+1)$, we conclude that $(i+1)$ stops waiting before the time we postulated for it to stop waiting.

Our experience suggests that a temporal argument, particularly if it employs proof by contradiction, can be replaced by a succinct formal argument using the **co**-properties.

A more involved example of mutual waiting arises when a process can wait for any one of a set of processes. This is the typical situation where processes hold resources and a waiting process can proceed only after receiving *any one* of the resources for which it is waiting. Figure 3.3 shows an example in which four processes, x, y, z, u, are deadlocked. The outgoing arrows from x (to y and z) denote that x is waiting for one of y and z.

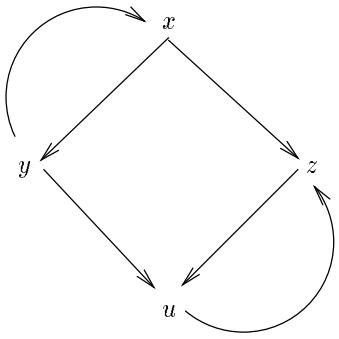


Figure 3.3: A Knot of Waiting Processes

The waiting condition for x—using xw, yw, zw to denote that x, y, z are waiting, respectively—is that $xw \wedge yw \wedge zw$ **co** xw.

Unlike the previous case, the existence of a cycle in the graph does not guarantee deadlock. A nonempty set of nodes, S, in a directed graph forms a *knot* if every outgoing edge from a node in S is to a node in S. Figure 3.3 has knots $\{x, y, z, u\}$ and $\{z, u\}$. We show that when all processes in a knot S are waiting in the above sense, they are deadlocked. As before, for a process i in S, w_i denotes that process i is waiting. Let process i have outgoing edges to nodes in S_i, $S_i \subseteq S$. Then, the waiting condition for process i is given by

$$w_i \wedge \langle \forall\ j\ :\ j \in S_i\ :\ w_j \rangle \ \textbf{co}\ w_i$$

Taking the conjunction over all i in S

$$\langle \forall\ i\ :\ i \in S\ :\ w_i \wedge \langle \forall\ j\ :\ j \in S_i\ :\ w_j \rangle \rangle \ \textbf{co}\ \langle \forall\ i\ :\ i \in S\ :\ w_i \rangle$$

Simplifying the lhs, using $S_i \subseteq S$,

$$\langle \forall\ i\ :\ i \in S\ :\ w_i \rangle \text{ stable}$$

Observe that the type of waiting is captured by the **co**-property for each i and the structure of the knot is exploited in simplifying the conjunction. An informal proof, we suspect, will be much longer.

3.5.8 Axiomatization of a Communication Network

In this example, we develop axioms to describe a message communicating network. These axioms are based on Misra[20, Section 6.1] . A more general case, based on Chandy and Misra[4], is treated in Exercise 16.

To keep matters simple, suppose that we have two processes, A, B, that communicate via two one way channels. The number of messages sent by a process is at least as large as the number received by the other, and both quantities are nonnegative. Furthermore, the number of messages sent and received along each channel is nondecreasing. A channel is *empty* if the number of messages sent equals the number of messages received along that channel. The state of a process is either *idle* or *active*. An idle process remains idle until it receives a message. Only active processes may send messages.

Our main interest is in specifying the properties of the system, formally. We will also show that when both processes are idle and both channels are empty then the system is terminated, in the sense that no further messages will be sent nor received nor will any of the processes become active.

In describing a system, we first have to decide on the variables whose values determine the state of the system. The choice is not always clear. In this case, for instance, we will not explicitly introduce a channel state. Instead, we introduce sA, sB to denote the number of messages sent by A, B and rA, rB for the number of messages received by A, B, respectively. The channel from A to B, for example, is *empty* if $sA = rB$. That the number of messages sent by a process is at least as large as the number received by the other, and both quantities are nonnegative, is given by

$$\begin{array}{l} sA \geq rB \geq 0 \text{ invariant} \\ sB \geq rA \geq 0 \text{ invariant} \end{array} \qquad \text{(CN1)}$$

The number of messages sent and received are nondecreasing. Using free variables m, n (of type natural)

$$\begin{array}{l} sA \geq m \text{ stable}, \quad sB \geq m \text{ stable}, \\ rA \geq n \text{ stable}, \quad rB \geq n \text{ stable} \end{array} \qquad \text{(CN2)}$$

To express the facts dealing with the states of a process we introduce the boolean variable qA denoting that A is idle; similarly, qB.

The following property says that each process remains idle as long as it receives no message.

$$\begin{array}{l} qA \ \wedge \ rA = m \ \mathbf{co} \ rA = m \ \Rightarrow \ qA \\ qB \ \wedge \ rB = m \ \mathbf{co} \ rB = m \ \Rightarrow \ qB \end{array} \qquad \text{(CN3)}$$

Note that it is not specified if an idle process becomes active upon receiving a message; our specification allows it to be in either state. That only active processes may send messages is best understood as, an idle process does not send messages, i.e.,

$$qA \ \wedge \ sA = n \ \textbf{co} \ sA = n$$
$$qB \ \wedge \ sB = n \ \textbf{co} \ sB = n \qquad \text{(CN4)}$$

A Digression: There is a subtle point in our formulation of CN4. We prohibit an idle process from receiving a message, becoming active and then sending a message, all in one step. If we wished to allow this possibility we would have written (CN4) analogously to CN3:

$$qA \ \wedge \ sA = n \ \textbf{co} \ sA = n \ \Rightarrow \ qA$$
$$qB \ \wedge \ sB = n \ \textbf{co} \ sB = n \ \Rightarrow \ qB$$

Observe that with this specification we can no longer prove termination: The specification does not say that an idle process has to receive a message *strictly* prior to becoming nonidle and sending a message. Therefore, in a state where both processes are idle and both channels are empty, a next state is possible where both processes are idle, both channels are empty *and* one additional message has been sent and received along every channel. □

This completes our axiomatization. Our axioms capture only some aspects of the system behavior. In particular, we have no guarantee that all messages are eventually delivered—a progress property—or that messages are delivered in the same order in which they are sent (a safety property). But it should be clear how we intend to proceed in order to state such properties. Exercise 16 asks you to write the axioms analogous to CN1–CN4 for an arbitrary network.

The state in which both processes are *idle* and both channels are *empty* is given by

$$(qA \ \wedge \ qB) \ \wedge \ sA, sB = rB, rA$$

We can show that the above predicate is stable. But that is not enough; it leaves open the possibility, for instance, that the values of sA, rB could change while preserving this predicate. Therefore, we will prove that once the above predicate holds none of the variables will change value. Since $sA, sB = rB, rA$ from the predicate, it is sufficient to show that rA, rB do not change value, i.e., for free m, n

$$(qA \ \wedge \ qB) \ \wedge \ sA, sB = rB, rA \ \wedge \ rA, rB = m, n \text{ stable} \qquad \text{(CN5)}$$

Proof of CN5:

We repeat CN3, CN4 for A.

$$qA \ \wedge \ rA = m \ \textbf{co} \ rA = m \ \Rightarrow \ qA \qquad \text{(CN3)}$$
$$qA \ \wedge \ sA = n \ \textbf{co} \ sA = n \qquad \text{(CN4)}$$

Conjoining the above two

$$qA \ \wedge \ rA, sA = m, n \ \mathbf{co} \ (rA = m \ \Rightarrow \ qA) \ \wedge \ sA = n$$

Similarly, for B, using n, m in place of m, n

$$qB \ \wedge \ rB, sB = n, m \ \mathbf{co} \ (rB = n \ \Rightarrow \ qB) \ \wedge \ sB = m$$

Conjoining the above two

$$\begin{array}{l} (qA \ \wedge \ qB) \ \wedge \ rA, sA = m, n \ \wedge \ rB, sB = n, m \\ \qquad\qquad \mathbf{co} \\ (rA = m \ \Rightarrow \ qA) \ \wedge \ (rB = n \ \Rightarrow \ qB) \ \wedge \ sA, sB = n, m \end{array} \qquad \text{(CN6)}$$

From CN2,

$rA \geq m$ stable , $rB \geq n$ stable

Therefore, their conjunction is stable. Conjoining this to both sides of CN6 (and rewriting its lhs and rhs)

$$\begin{array}{l} (qA \ \wedge \ qB) \ \wedge \ rA, rB = sB, sA \ \wedge \ rA, rB = m, n \\ \qquad\qquad\qquad\qquad\qquad \mathbf{co} \\ (rA, rB = m, n \ \Rightarrow \ qA \ \wedge \ qB) \ \wedge \ sA, sB = n, m \ \wedge \\ \qquad rA \geq m \ \wedge \ rB \geq n \end{array}$$

From CN1, $sA \geq rB$ and $sB \geq rA$. Using (CN1) in the rhs, we get $rA, rB = m, n$ in the rhs. Simplifying the rhs, it is seen to be identical to the lhs.

3.5.9 Coordinated Attack

The following vicious version of a synchronization problem has received considerable attention in the literature. We show how it can be dealt with, relatively simply, by using our theory. The following description of the problem is from Gray[9].

> "Two divisions of an army are camped on two hilltops overlooking a common valley. In the valley awaits the enemy. It is clear that if both divisions attack the enemy simultaneously they will win the battle, whereas if only one division attacks it will be

> defeated. The divisions do not initially have plans for launching an attack on the enemy, and the commanding general of the first division wishes to coordinate a simultaneous attack (at some time the next day). Neither general will decide to attack unless he is sure that the other will attack with him. The generals can only communicate by means of a messenger. Normally, it takes the messenger one hour to get from one encampment to the other. However, it is possible that he will get lost in the dark or, worse yet, be captured by the enemy. Fortunately, on this particular night, everything goes smoothly. How long will it take them to coordinate an attack?"

The original arguments to show that there is no protocol to achieve coordinated attack were long and cumbersome. Halpern and Moses[10] gave the first convincing proof of this fact based on the notions of *knowledge* and *common knowledge*. We give a very brief outline of that theory as it pertains to this problem.

For a process x and predicate p, $x\ k\ p$ denotes that "x knows p". The value of the predicate $x\ k\ p$ is a function of the state of x; hence $x\ k\ p$ does not change as long as x performs no transition (a transition is either a local computation, sending a message or receiving a message). For a predicate p, predicate cp denotes that p is *common knowledge* (for some group of processes). We have, for any process x in the group

$$cp \ \equiv \ x\ k\ (cp)$$

That is, p is common knowledge iff all processes know it to be common knowledge.

The coordinated attack problem requires us to establish common knowledge of the attack time. Halpern and Moses[10] showed that if any atomic transition affects the state of no more than one process, as is customarily the case in asynchronous message communicating systems, then common knowledge cannot be gained. Later, Chandy and Misra[2] showed that under the same assumptions, common knowledge can neither be gained nor lost; that is, a previously unplanned attack cannot be coordinated nor can a planned coordinated attack be called off. This result holds under the much weaker assumption that no transition can affect the states of *all* processes (though any proper subset of processes could be affected). We prove this more general result below.

Theorem: In a group having more than one process where no atomic transition can affect the states of *all* processes, common knowledge can neither be gained nor lost; that is, for any predicate p,

cp constant

Proof: An atomic transition does not affect the states—hence the knowledge predicates—of *all* processes simultaneously. Therefore, in a group having more than one process, knowledge predicate of at least one process is unchanged by any transition. As in Example (9) of Section 3.2, we write this assertion for any predicate q—where x_i denotes the i^{th} process, i is quantified over all processes in the group and b is a free boolean variable

$$\langle \forall\ i \quad :: \quad (x_i\ k\ q) = b \rangle \ \ \mathbf{co} \ \ \langle \exists\ i \quad :: \quad (x_i\ k\ q) = b \rangle$$

Instantiate q by cp. Then, $x_i\ k\ q = x_i\ k\ (cp) = cp$.

$\langle \forall\ i \quad :: \quad cp = b \rangle \ \ \mathbf{co} \ \ \langle \exists\ i \quad :: \quad cp = b \rangle$	
$cp = b \ \mathbf{co}\ cp = b$	, i is quantified over a nonempty group
cp constant	, definition of constant

3.5.10 Dynamic Graphs

This example, based on Chandy and Misra[3, Chapter 12] and Misra[19], illustrates how we express and deduce facts about dynamic data structures. In this example, the data structure is a finite directed graph which can be changed by the following operation: All edges incident on a node may be directed towards that node in one (atomic) step.

A node that has no outgoing edge is called *sink*. We are required to show that no path is ever created to a nonsink, that is if there is no path from node u to node v initially then there is no path from u to v at any point in the computation unless v is a sink at that point.

Let us first give a typical proof that draws upon the well known theorems and terminology of graph theory. Suppose that there is no path from u to v before a step. Call all the edges of the graph "old" at this time. Following the step that redirects all incident edges towards a node, w, call the newly redirected edges to w "new." Suppose there is a path from u to v following the step and v is nonsink. Not all edges on the path are "old" because then there would have been a path before the step. Since some new edge is on this path, node w is on this path. We show that node w is not on the path, thus leading to a contradiction: Node w is different from v because v is assumed to be a nonsink node, and w is a sink node after the step; node w is not an intermediate node on the path nor is $w = u$, because w has no outgoing edge.

Our formal proof avoids the temporal argument and the proof by contradiction. However, the proof has to make explicit the notion of a path. In the following proof, u, v, w range over nodes of the graph. We use the following predicates:

$u.-$: u is a sink
$u\ R^k\ v,\ k > 0$: there is a path of length k from u to v
$u\ R\ v$: there is a path from u to v

Now, $u\ R^1\ v$ simply denotes that there is an edge from u to v; we define $u\ R^{k+1}\ v$, $k \geq 1$, inductively as follows.

$$u\ R^{k+1}\ v \quad \equiv \quad \langle \exists\ w \quad :: \quad u\ R^1\ w\ \wedge\ w\ R^k\ v \rangle \tag{GR1}$$

Then,

$$u\ R\ v \quad \equiv \quad \langle \exists\ k\ :\ k \geq 1\ :\ u\ R^k\ v \rangle \text{ and} \tag{GR2}$$
$$u.- \quad \equiv \quad \langle \forall\ v\ :\ \neg u\ R\ v \rangle \tag{GR3}$$

The operation of changing the graph is given by the following **co**-property. It states that no edge from u to v is created as long as v remains a nonsink.

$$\neg u\ R^1\ v\ \ \mathbf{co}\ \ \neg u\ R^1\ v\ \vee\ v.- \tag{GR4}$$

We prove that no path from u to v is created as long as v remains a nonsink.

Theorem: $\neg u\ R\ v\ \ \mathbf{co}\ \ \neg u\ R\ v\ \vee\ v.-$

Proof: We show that for all k, $k \geq 1$, and all u, v

$$\neg u\ R^k\ v\ \ \mathbf{co}\ \ \neg u\ R^k\ v\ \vee\ v.- \tag{GR5}$$

Taking conjunction of (GR5) over all k, $k \geq 1$, concludes the proof of the theorem.

Proof of (GR5): by induction on k.

$k = 1$: the result follows from (GR4)
$k + 1$: show $\neg u\ R^{k+1}\ v\ \ \mathbf{co}\ \ \neg u\ R^{k+1}\ v\ \vee\ v.-$

From (GR4), using w for v, we have

$\neg u\ R^1\ w$ **co** $\neg u\ R^1\ w\ \vee\ w.-$

Using induction hypothesis (GR5) with w in place of u we have

$\neg w\ R^k\ v$ **co** $\neg w\ R^k\ v\ \vee\ v.-$

Disjunction of the above two gives us

$\neg u\ R^1\ w\ \vee\ \neg w\ R^k\ v$ **co** $\neg u\ R^1\ w\ \vee\ \neg w\ R^k\ v\ \vee\ w.-\ \vee\ v.-$

From (GR3) $w.- \Rightarrow\ \neg w\ R\ v$ and from (GR2) $\neg w\ R\ v\ \Rightarrow\ \neg w\ R^k\ v$. Hence, $\neg w\ R^k\ v \vee w.-$ in the rhs of the above can be replaced by $\neg w\ R^k\ v$.

$\neg u\ R^1\ w\ \vee\ \neg w\ R^k\ v$ **co** $\neg u\ R^1\ w\ \vee\ \neg w\ R^k\ v\ \vee\ v.-$

Taking conjunction of the above over all w,

$\langle \forall\ w\ ::\ \neg u\ R^1\ w\ \vee\ \neg w\ R^k\ v\rangle$ **co**
$\langle \forall\ w\ ::\ \neg u\ R^1\ w\ \vee\ \neg w\ R^k\ v\rangle\ \vee\ v.-$

Using (GR1), replace $\langle \forall\ w\ ::\ \neg u\ R^1\ w\ \vee\ \neg w\ R^k\ v\rangle$ by $\neg u\ R^{k+1}\ v$

$\neg u\ R^{k+1}\ v$ **co** $\neg u\ R^{k+1}\ v\ \vee\ v.-$. □

We note, as a corollary, that once a node u is outside any cycle, it remains outside cycles. In particular, once the graph becomes acyclic, it stays acyclic. Node u is outside any cycle if $\neg u\ R\ u$ holds.

Corollary: $\neg u\ R\ u$ stable.

Proof:

$\neg u\ R\ v$ **co** $\neg u\ R\ v\ \vee\ v.-$	, theorem
$\neg u\ R\ u$ **co** $\neg u\ R\ u\ \vee\ u.-$	, replacing v by u in the above
$\neg u\ R\ u$ **co** $\neg u\ R\ u\ \vee\ \neg u\ R\ u$	, weakening rhs using (GR3) for $u.- \Rightarrow\ \neg u\ R\ u$
$\neg u\ R\ u$ stable	, predicate calculus on the rhs

3.6 Theoretical Results

There are several results about **co** that are used in formulating other concepts, such as the strongest invariant or FP; however, these results are rarely used in dealing with specific applications.

3.6.1 Strongest rhs; weakest lhs

In a given program, for any predicate p there is a strongest q such that p **co** q; similarly, for any q there is a weakest p such that p **co** q. We only prove the first result; the proof of the second one is similar.

Theorem (strongest rhs): For any p there exists a q such that

- p **co** q and,
- $(\forall\ r\ :\ p\ \mathbf{co}\ r\ :\ q\ \Rightarrow\ r)$

Proof: We give an explicit description of q; it is the conjunction of the rhs of all **co**-properties in which p appears in the lhs:

$$q\ \equiv\ \langle \wedge\ b\ :\ p\ \mathbf{co}\ b\ :\ b \rangle$$

Since **co** is universally conjunctive, we have p **co** q. For any r such that p **co** r, we get $q\ \Rightarrow\ r$ from the definition. □

3.6.2 Strongest Invariant

Every program has a (unique) strongest invariant. This is proved by defining the strongest invariant to be, simply, the conjunction of all invariants. We give a longer alternative proof which provides a "constructive" procedure for obtaining the strongest invariant. The essence of the procedure is to start with the initial condition—call it p_0, and obtain a sequence of p_is where p_i **co** p_{i+1} and p_{i+1} is the strongest rhs for p_i. The disjunction of all the p_is is the strongest invariant.

Lemma: Let $p_0, \ldots, p_i, \ldots$ be an infinite sequence where, for all i, $i \geq 0$,

$$p_i\ \mathbf{co}\ p_{i+1}$$

Then, $\langle \exists\ i\ ::\ p_i \rangle$ stable.

Proof: Taking the disjunction over all the **co**-properties, p_i **co** p_{i+1}, we get

$$\langle \exists\ i\ :\ i \geq 0\ :\ p_i \rangle \textbf{ co } \langle \exists\ i\ :\ i > 0\ :\ p_i \rangle$$

Weakening the rhs by p_0, we obtain the result. □

Theorem (strongest invariant): Let p_0 be the initial condition. Let p_i **co** p_{i+1}, for all i, $i \geq 0$, where p_{i+1} is the strongest rhs for p_i. Then, $\langle \exists\ i\ ::\ p_i \rangle$ is the strongest invariant.

Proof: Let $P = \langle \exists\ i\ ::\ p_i \rangle$. From the previous lemma, P is stable. Furthermore, since $p_0 \Rightarrow P$, initially P holds. Therefore, P is an invariant.

To show that P is the strongest invariant, we show that for any invariant J, $P \Rightarrow J$, i.e., $\langle \forall\ i\ ::\ p_i \Rightarrow J \rangle$. The proof is by induction on i.

$p_0 \Rightarrow J$: initially J holds. Since p_0 is the initial condition, $p_0 \Rightarrow J$.

Assume $p_i \Rightarrow J$ and show $p_{i+1} \Rightarrow J$:

p_i **co** p_{i+1}	, given
$p_i \wedge J$ **co** $p_{i+1} \wedge J$	, stable conjunction with J
p_i **co** $p_{i+1} \wedge J$	, $p_i \wedge J \equiv p_i$ since $p_i \Rightarrow J$
$p_{i+1} \Rightarrow p_{i+1} \wedge J$	, p_{i+1} is the strongest rhs in any **co**-property whose lhs is p_i
$p_{i+1} \Rightarrow J$	, predicate calculus

□

The strongest invariant includes the initial condition, p_0, as a disjunct. Therefore, if p_0 is not identically *false* then neither is the strongest invariant. Conversely, if p_0 is *false* then so is p_1—because, *false* is stable—and hence, the strongest invariant is *false*. We have often used the term "reachable states" informally in this chapter. Formally, a state is *reachable* iff it satisfies the strongest invariant.

3.6.3 Fixed Point

The predicate FP characterizes the set of states which do not change as a result of program execution. We can define FP using **co**. Observe that any subset of states satisfying FP is stable, i.e., for any predicate b

$$FP \wedge b \quad \text{stable}$$

This, however, does not identify a unique FP. In particular, *false* $\wedge$ b sis table, for any b. We define FP to be the *weakest* predicate p such that

$p \wedge b$ is stable, for all b. It can be shown that FP has the following closed form (see Exercise 14).

$$FP \equiv \langle \exists\, p \;:\; \langle \forall\, b \;\; :: \;\; p \wedge b \;\; \text{stable} \rangle \;:\; p \rangle$$

As is the case for the strongest invariant, it is difficult to compute FP using the above formulation; we have shown earlier how to obtain FP by syntactic manipulations of the program text.

A program is "deadlock-free" iff FP is always-*false*, i.e., $\neg FP$ is always-*true*. Using the substitution axiom, then $\neg FP$ is invariant.

3.7 Synopsis

This section contains an assessment of the strengths and weaknesses of **co**. Earlier, we had gained considerable experience in writing and manipulating *unless* properties[3, 25], a predecessor of **co**. There is ample reason to believe that **co** would be at least as powerful for expressing the usual kinds of safety properties for reactive systems. The manipulation rules for **co** are simple and effective. The examples in Section 3.5—particularly, common meeting time (3.5.2), deadlock (3.5.7) and coordinated attack (3.5.9)—were handled by extremely concise proofs; it is difficult to see how intuitive reasoning could be any cheaper! Though our examples often used UNITY-style programs, the theory is applicable to any transition system.

The proposed theory has several limitations. The first concerns expressibility. There are safety properties that cannot be expressed using **co**, or that can be expressed only with some difficulty. A property that cannot be expressed is: For each value, n, between 0 and 9, there exists a finite execution of the program so that x has value n at the end of the execution. This kind of property is succinctly expressed using branching time temporal logic. Our theory, based on linear temporal logic, is inadequate in such cases. This limitation, though, is deliberate; we have tried to avoid elaborate theories, and branching time logics would require such theories.

Temporal logic is an elegant extension of classical logic that includes the temporal operators, $\Box$ and $\Diamond$ (read as *always* and *eventually*). The primary operator for expressing the safety properties is $\Box$; for example, the temporal formula $p \;\Rightarrow\; \Box p$ denotes that once p is true it is always true, i.e., p is stable. Temporal logic has a well developed theory and it has been applied in a variety of problems in computer science, see Manna and Pnueli[18]. There are many properties that are easily expressible in linear temporal logic but are difficult to express using **co**. Consider Exercise (9e): Once x exceeds 5 it remains positive. In temporal logic, this is simply

$$x > 5 \;\Rightarrow\; \Box\,(x > 0).$$

Such a property cannot be directly written using **co** because examining a system state, say, $x = 2$, yields little clue about the value of x in the next state. Whether $x > 0$ in some state may depend on the history of the computation; thus, for any state we have to know if x has exceeded 5 in the past. Introducing an auxiliary boolean variable b that is true iff x has ever exceeded 5, we have

initially $b \;\equiv\; x > 5$,
$x > 5 \;\Rightarrow\; b$,
$b \;\wedge\; x > 0$ stable

Note, however, that in temporal logic the proof of

$$x > 5 \;\Rightarrow\; \Box\,(x > 0)$$

would require the introduction of a predicate analogous to b.

Event predicates[14] and TLA[17] have proved useful for describing safety properties. An example of an event predicate is $x' \geq x$ which says that the value of x after any transition—this value is denoted by x'—is at least the value of x before the transition—the value before the transition is, simply, denoted by x. This formalism is attractive because the inference rules are simply those of predicate calculus. Lamport combines event predicates, temporal operators and quantification (over variables and actions) to obtain a powerful logic, called TLA. He advocates using the same logic for representing both a system and its properties, thus simplifying the proofs of implementations.

A disadvantage of our theory (compared to event predicates and TLA) is that we often have to introduce free variables in stating the properties. For instance, "x is nondecreasing" is written as

$$x \geq m \text{ stable}$$

with a free variable m, whereas

$$x' \geq x$$

is an event predicate that expresses the same fact. In the other parts of our theory, free variables are essential (to say, for instance, that x increases eventually). Therefore, we have introduced free variables also in this part of the theory dealing with safety.

3.8 Bibliography

The notions of pre- and post-conditions are from Floyd[8] and Hoare[11]. The *wp*-calculus is due to Dijkstra[6]; see Dijkstra and Scholten[7] for an elaborate treatment of predicate transformers and their applications in program semantics. Lamport[15] was the first to coin the terms *safety* and *liveness*; formal definitions of these terms are in Alpern and Schneider[1]. Our earlier theory of safety[3], using the *unless* operator, was inspired by temporal logic. The present treatment tries to overcome some of the pragmatic difficulties of using *unless*. There are a number of papers on the substitution axiom, in particular by Sanders[24], Knapp[13] and Misra[22]; the interpretation given in this chapter is from Knapp. A clear example of the distinction between "invariant" and "always-true" is in van Gasteren and Tel[27]. The notion of the strongest invariant has been around for a long time; see Lamport[16] and Sanders[24], in particular. Section 3.5.6, Auxiliary Variables, follows the treatment in Misra[21]. For completeness of UNITY logic see Jutla, Knapp and Rao[12] and Cohen[5]. Exercise 18 is from the latter reference. There are many other formalisms that are effective for expressing safety properties. Notable among these are temporal logic[18], event predicates[14] and TLA[17], which have been discussed in Section 3.7. Rao[23] has proposed logical operators for expressing "probabilistic" safety properties.

3.9 Exercises

1. (Formal Manipulation) Prove the following.

   (a) $$\frac{p \ \wedge \ q \ \mathbf{co} \ p \ , \ \neg q \quad \text{stable}}{p \ \mathbf{co} \ p \ \vee \ \neg q}$$

   (b) $$\frac{p \ \mathbf{co} \ q \ , \ \neg p \ \mathbf{co} \ \neg q}{p \quad \text{constant}}$$

   (c) (cancellation) $$\frac{p \ \mathbf{co} \ q \ \vee \ r \ , \ q \ \mathbf{co} \ b}{p \ \mathbf{co} \ b \ \vee \ r}$$

   (d) $$\frac{p \ \wedge \ \neg FP \ \mathbf{co} \ p}{p \quad \text{stable}}$$

, where FP is the fixed point of the given program

(e)
$$\frac{I \quad \text{invariant} \ , \ I \ \mathbf{co} \ p}{p \quad \text{stable}}$$

Hint: Use the substitution axiom.

(f)
$$\frac{\neg p \ \mathbf{co} \ p}{p \ \text{invariant}}$$

(g)
$$3 \ \text{constant}$$

(h)
$$\frac{x + y \ \text{constant} \ , \ y \ \text{constant}}{x \ \text{constant}}$$

(i) In the following p_i is a predicate where i ranges over 0 through $N-1$, and $N \geq 1$. We write i' to denote $(i+1) \bmod N$.

$$\frac{p_i \ \wedge \ \neg p_{i'} \ \mathbf{co} \ p_i \ \vee \ p_{i'}}{\langle \exists \ i \quad :: \quad p_i \rangle \ \wedge \ \langle \exists \ i \quad :: \quad \neg p_i \rangle \ \mathbf{co} \ \langle \exists \ i \quad :: \quad p_i \rangle}$$

2. (Manipulation of Sets) Show

   (a) For a set g, any constant set G, free variables x and predicates p, q that do not name x

   $$\frac{p \ \wedge \ x \in g \ \mathbf{co} \ x \in g \ \vee \ q}{p \ \wedge \ g \supseteq G \ \mathbf{co} \ g \supseteq G \ \vee \ q}$$

   (b)

   $$\frac{p \ \wedge \ x \notin g \ \mathbf{co} \ x \notin g \ \vee \ q}{p \ \wedge \ g \subseteq G \ \mathbf{co} \ g \subseteq \ G \ \vee \ q}$$

   Hint: Note that $g \subseteq G \ \equiv \ \langle \forall \ x \quad :: \quad x \in g \ \Rightarrow \ x \in G \rangle$

3. (Proving By Parts) Let $p(x,y), q(x,y)$ be predicates that name program variables x, y only. In order to prove

   $$p(x,y) \ \mathbf{co} \ q(x,y)$$

   the following strategy is suggested: for free variables m, n prove

$p(x,n)$ **co** $q(x,n)$ and
$p(m,y)$ **co** $q(m,y)$

Show that this is not a valid strategy. Next, show under the additional assumption that x, y are never changed simultaneously, the desired result can be proven.

4. (Substitution Axiom) Show that in the program *distinction* of Section 3.3.1, y constant .

5. (Substitution Axiom) Given that $x = y$ invariant holds in a program show that x can be replaced by y in any **co**-property. This is the Leibniz principle for a program.

6. (Proving From Program Text) For the following program, *Alternate*, show that,

   (a) x remains unchanged as long as it differs from y, i.e., for any m,

   $$x = m \ \wedge \ x \neq y \ \textbf{co} \ x = m$$

   (b)

   $$0 \leq x - y \leq 1 \text{ invariant}$$

   (c) the program is deadlock-free

   Program *Alternate*

   var x, y : natural
   initially $x, y = 1, 0$
   assign
   $x := x + 1$ if $x = y$
   $[\!]$ $y := y + 1$ if $x \neq y$
   end

7. (Elimination Theorem) Prove, for integer program variables x, y and free variables m, n:

   (a) $(x = m \ \textbf{co} \ x \geq m) \ \equiv \ (x \geq n \text{ stable})$

(b)
$$\frac{x, y = m, n \ \textbf{co}\ x, y = m, n \ \vee \ (m > n \ \wedge \ x, y = m - 1, n)}{x \geq y \quad \text{stable}}$$

(c)
$$\frac{x, y = m, n \ \textbf{co}\ x, y = m, n \ \vee \ x, y = m + 1, n - 1}{x + y \quad \text{constant}}$$

(d) If x is nondecreasing and y is nonincreasing then $x \geq y$ stable.

(e) If set s is nongrowing then its size is nonincreasing.

(f)
$$\frac{\begin{array}{c} x = m \ \textbf{co}\ p[x \ := \ m] \ \Rightarrow \ p \\ p \text{ does not name } m \text{ nor any program variable other than } x \end{array}}{p \text{ stable}}$$

8. (Elimination Theorem) Let f be a function that does not name m nor any program variable other than x. Show, for any m, n

(a)
$$\frac{\begin{array}{c} x = m \ \textbf{co}\ f.x \ \sim \ f.m \\ \sim \text{ transitive} \end{array}}{f.x \ \sim \ n \quad \text{stable}}$$

(b)
$$\frac{x = m \ \textbf{co}\ f.x = f.m}{f.x \quad \text{constant}}$$

(c) A function, f, preserves a relation, $\sim$, means $x \ \sim \ y \ \Rightarrow \ f.x \ \sim \ f.y$

$$\frac{\begin{array}{c} x = m \ \textbf{co}\ x \ \sim \ m \\ f \text{ preserves } \sim \\ f \text{ does not name } m \text{ nor any program variable other than } x \end{array}}{f.x = n \ \textbf{co}\ f.x \ \sim \ n}$$

9. (From Prose to Formula) Convert the following verbal descriptions into formal properties and then establish the conclusions wherever suggested. Note that, like most verbal descriptions, each of the following admits of several possible interpretations. Experiment!

   (a) Predicates p, q change synchronously. Hint: There is a neat way to write this.

   (b) For an integer variable x let sx be an integer variable that is increased by the new value of x whenever x is changed. Variable sx is not changed otherwise. Show that if x is always nonnegative then $sx \geq x$ stable.

   (c) Let x, y be integer variables. Let c be an auxiliary variable that counts the number of assignments to x or y that causes x to exceed y. Show that if $x \leq y$ is invariant then $c = 0$ is stable.

   (d) Formalize: Once integer x exceeds 5 it never decreases.

   (e) Formalize: Once integer x exceeds 5 it remains positive forever. Hint: Introduce an auxiliary predicate.

   (f) A vertex (in a directed graph) remains *black* until it has a *non-black* neighbor. Show that the state where all vertices are black persists.

10. (More Prose to Formula) An impermeable vessel consists of chambers numbered 0 through N. The particles inside the vessel move according to the following law: A particle may stay in its current chamber or move to a higher numbered chamber. Impermeability of the vessel guarantees that particles cannot enter or leave the vessel. Show that

    (a) The set of particles in the chambers at or above j, $0 \leq j \leq N$, is nonshrinking.

    (b) The set of particles in the chambers at or below j, $0 \leq j \leq N$, is nongrowing.

    (c) The set of particles in the vessel is constant.

11. (Methodology) The following methodology is suggested to express all the safety properties of a program. For each variable or groups of variables x, write

$$x \ : \ p \ \rightarrow \ q$$

    to denote that (in a given program) variable x can change only if p is a precondition, and every change in x guarantees q as a postcondition.

Express these requirements using **co**.

Show that

(a)

$$\frac{x \ : \ p \ \rightarrow \ q}{x \ : \ p \ \vee \ p' \ \rightarrow \ q \ \vee \ q'}$$

(b)

$$\frac{x \ : \ p \ \rightarrow \ q \ , \ x \ : \ p' \ \rightarrow \ q'}{x \ : \ p \ \wedge \ p' \ \rightarrow \ q \ \wedge \ q'}$$

12. (Fixed Point) Show programs with program variable x (of type integer) whose FPs are

    (a) *true*

    (b) $x \neq 0$

    (c) $x \leq 0$

    (d) $\langle \exists \ n \ :: \ x^n = 64 \rangle$. Here n is a positive integer.

13. (Fixed Point) A set of numbers are arranged cyclically. A *move* consists of replacing a number x by $|x - x'|$ where x' is the right neighbor of x. Show that some number can be changed iff there is a nonzero number.

    Note: It is not guaranteed that eventually all numbers will become zero if moves are applied repeatedly. The following variation is an interesting puzzle: If *all* numbers are replaced simultaneously (by applying moves to all of them) and there are N numbers where N is a power of 2, then all numbers will eventually become zero.

14. (Closed Form For Fixed Point) In this exercise, x, b, p, FP are predicates. Define FP by $FP \equiv \langle \exists \ p \ : \ \langle \forall \ b \ :: \ p \ \wedge \ b \ \text{ stable} \rangle \ : \ p \rangle$. Show that FP is the weakest solution (in x) to

    (a) $\langle \forall \ b \ :: \ x \ \wedge \ b \ \text{ stable} \rangle$

15. (Finite State Descriptions) There are two communicating processes, P, Q; any message sent by P is acknowledged by Q. In this exercise we consider the transfer of a single message and the corresponding acknowledgement. The state of the system is given by two boolean

variables, p, q. Initially, both variables are *false*. Variable p is set *true* by P when it sends a message, and set *false* when it receives an acknowledgement. Variable q is set *true* when Q receives the message; it remains *true* thereafter.

Draw a state transition diagram. Construct the **co**-properties. Obtain an equivalent set of **co**-properties in which a single literal ($p, q, \neg p$ or $\neg q$) appears in the lhs.

16. (Axiomatizing Communication Networks) Consider a ring network consisting of at least two processes. Develop a formalization of its properties along the same lines as in Section 3.5.8.

    Suggested Notation: Let i' denote the process to which process i sends messages.

    Provide a similar formalization of an arbitrary network.

    Suggested Notation: For a channel c, let $s.c, r.c$ denote the number of messages sent and received along c. For sets of processes x, y use xy to denote the set of channels from a process in x to a process in y. Use the following abbreviations

    $$(s = r).xy \;\equiv\; \langle \forall\ c\ :\ c\ \in\ xy\ :\ s.c = r.c \rangle$$
    $$(s \geq r).xy \;\equiv\; \langle \forall\ c\ :\ c\ \in\ xy\ :\ s.c \geq r.c \rangle$$

    Similarly, $(s = L).xy$ or $(r = L).xy$ for free L are defined. In both cases, prove results analogous to (CN5).

17. Show that for any predicate p there exists (in a given program) a strongest stable predicate weaker than p. Denoting this predicate by $ss.p$, show that

    (a) ss is monotonic
    $(p \;\Rightarrow\; q) \;\Rightarrow\; (ss.p \;\Rightarrow\; ss.q)$

    (b) p stable $\equiv\ (ss.p = p)$

    (c) ss is idempotent
    $ss.(ss.p) = ss.p$

    (d) ss is universally disjunctive
    $ss.\langle \exists\ i\ \ ::\ \ p_i \rangle = \langle \exists\ i\ \ ::\ \ ss.p_i \rangle$

    Deduce similar facts about the weakest stable predicate stronger than p.

Answers to Exercises

1. (a)
   $p \wedge q$ **co** p , given
   $\neg q$ stable , given
   $p \vee \neg q$ **co** $p \vee \neg q$, disjunction of the above two
   p **co** $p \vee \neg q$, strengthening lhs of the above

   (b)
   $p \Rightarrow q$, from p **co** q
   $\neg p \Rightarrow \neg q$, from $\neg p$ **co** $\neg q$
   $p \equiv q$, from the above two
   p **co** p , from p **co** q and $p \equiv q$
   $\neg p$ **co** $\neg p$, from $\neg p$ **co** $\neg q$ and $p \equiv q$
   p constant , from the above two and definition of constant

   (c)
   $q \Rightarrow b$, from q **co** b
   p **co** $q \vee r$, given
   p **co** $b \vee r$, weakening rhs using $q \Rightarrow b$

   (d)
   $p \wedge \neg FP$ **co** p , given
   $p \wedge FP$ stable , stability at fixed point
   p **co** p , disjunction of the above two
   p stable , rewriting the above

   (e)
   I **co** p , given
   $I \wedge p$ **co** p , strengthening the lhs
   p **co** p , substitution axiom—I replaced by *true*—in the lhs

   (f)
   $\neg p$ **co** p , given
   $\neg p \Rightarrow p$, from the above
   p , simplifying
   p invariant , substitution axiom

   (g) We have to show, for all m,
   $3 = m$ stable
   Now,
   $\langle \forall\ m\ :\ m \neq 3\ :\ 3 = m$ stable$\rangle$
   , above predicate is *false* and *false* stable
   $\langle \forall\ m\ :\ m = 3\ :\ 3 = m$ stable$\rangle$
   , above predicate is *true* and *true* stable
   $\langle \forall\ m\ ::\ 3 = m$ stable$\rangle$
   , disjunction of the above two

   (h)
   y constant , given
   $x + y$ constant , given
   $(x + y) - y$ constant , constant formation rule

applied to the above two

x constant , from the above

(i) $p \wedge \neg p_{i'}$ **co** $p_i \vee p_{i'}$, given

$\langle \exists\ i \ :: \ p_i \wedge \neg p_{i'} \rangle$ **co** $\langle \exists\ i \ :: \ p_i \vee p_{i'} \rangle$
, disjunction over all i

$\langle \exists\ i \ :: \ p_i \rangle \wedge \langle \exists\ i \ :: \ \neg p_i \rangle$ **co** $\langle \exists\ i \ :: \ p_i \vee p_{i'} \rangle$
, $\langle \exists\ i \ :: \ p_i \wedge \neg p_{i'} \rangle \equiv \langle \exists\ i \ :: \ p_i \rangle \wedge \langle \exists\ i \ :: \ \neg p_i \rangle$

Result follows by simplifying the rhs.

2. (a) $p \wedge x \in g$ **co** $x \in g \vee q$
, given

$\langle \forall\ x \ : \ x \in G \ : \ p \wedge x \in g \rangle$ **co** $\langle \forall\ x \ : \ x \in G \ : \ x \in g \vee q \rangle$
, conjunction over all x, $x \in G$

$p \wedge g \supseteq G$ **co** $g \supseteq G \vee q$
, simplifying (because p, q do not name x)

(b) $p \wedge x \notin g$ **co** $x \notin g \vee q$
, given

$\langle \forall\ x \ : \ x \notin G \ : \ p \wedge x \notin g \rangle$ **co** $\langle \forall\ x \ : \ x \notin G \ : \ x \notin g \vee q \rangle$
, conjunction over all x, $x \notin G$

$p \wedge g \subseteq G$ **co** $g \subseteq G \vee q$
, simplifying

3. Consider a program consisting of integer variables x, y and a single transition

$$x, y \ := \ x + 1, y - 1$$

Let $p(x,y), q(x,y)$ be $x = y, x = y \vee x = y + 1$, respectively. For any m

$p(x,n)$ **co** $q(x,n)$
i.e., $x = n$ **co** $x = n \vee x = n + 1$ holds in the program.
Similarly, $p(m,y)$ **co** $q(m,y)$
i.e., $m = y$ **co** $m = y \vee m = y + 1$ also holds in the program.
However, $p(x,y)$ **co** $q(x,y)$ does not hold in this program.

Now, we prove the result assuming additionally that x, y are not changed simultaneously. That is,

$$\frac{\begin{array}{ccc} p(x,n) & \textbf{co} & q(x,n) \\ p(m,y) & \textbf{co} & q(m,y) \\ x, y = m, n & \textbf{co} & x = m \ \vee \ y = m \end{array}}{p(x,y) \quad \textbf{co} \quad q(x,y)}$$

Taking conjunction of the premises

$$p(x,m) \ \wedge \ p(m,y) \ \wedge \ x,y = m,n \ \mathbf{co} \ q(x,n) \ \wedge \ q(m,y) \\ \wedge \ (x = m \ \vee \ y = n)$$

Rewriting the lhs and weakening the rhs

$$p(x,y) \ \wedge \ x,y = m,n \ \mathbf{co} \ (q(m,y) \ \wedge \ x = m) \ \vee \\ (q(x,n) \ \wedge \ y = n)$$

Weakening the rhs

$$p(x,y) \ \wedge \ x,y = m,n \ \mathbf{co} \ q(x,y)$$

Taking the disjunction over all m, n

$$p(x,y) \ \mathbf{co} \ q(x,y)$$

4. The goal is to prove that for any m,

$$y = m \text{ stable}$$

We have,

$\langle \forall \ m \ : \ m \neq 0 \ : x = 0 \ \wedge \ y = 0 \ \wedge \ y = m \text{ stable} \rangle$
, *false* stable
$x = 0 \ \wedge \ y = 0 \ \wedge \ y = 0$ stable
, $x = 0 \ \wedge \ y = 0$ invariant
$\langle \forall \ m \ :: \ x = 0 \ \wedge \ y = 0 \ \wedge \ y = m \text{ stable} \rangle$
, from the above two
$\langle \forall \ m \ :: \ y = m \text{ stable} \rangle$
, substitution axiom: $x = 0 \ \wedge \ y = 0$ invariant

5. Let $p(x) \ \mathbf{co} \ q(x)$ denote a **co**-property in which $p(x), q(x)$ possibly mention x. Given that $x = y$ invariant we show $p(y) \ \mathbf{co} \ q(y)$.

$p(x) \ \mathbf{co} \ q(x)$, given
$p(x) \ \wedge \ x = y \ \mathbf{co} \ q(x) \ \wedge \ x = y$
, stable conjunction with $x = y$
$p(y) \ \wedge \ x = y \ \mathbf{co} \ q(y) \ \wedge \ x = y$
, $p(x) \ \wedge \ x = y \ \equiv \ p(y) \ \wedge \ x = y$ using Leibniz;
similarly, $q(x) \ \wedge \ x = y \ \equiv \ q(y) \ \wedge \ x = y$
$p(y) \ \mathbf{co} \ q(y)$, replace $x = y$ by *true* using substitution axiom

6. (a) Observe that $x = m \ \wedge \ x \neq y \ \Rightarrow \ x = m$. We show

$$\{x = m \ \wedge \ x \neq y\} \ \ x \ := \ x + 1 \qquad \text{if} \quad x = y \ \ \{x = m\}$$
$$\{x = m \ \wedge \ x \neq y\} \ \ y \ := \ y + 1 \qquad \text{if} \quad x \neq y \ \ \{x = m\}$$

These results follow from the axiom of assignment:

$$x = m \ \wedge \ x \neq y \ \wedge \ x = y \ \Rightarrow \ x + 1 = m \quad \text{and,}$$
$$x = m \ \wedge \ x \neq y \ \wedge \ x \neq y \ \Rightarrow \ x = m$$

(b) Prove this from the program text in a manner similar to 6a.

(c)
$$\begin{aligned} FP &= (x \neq y \ \vee \ x = x + 1) \ \wedge \ (x = y \ \vee \ y = y + 1) \\ &= x \neq y \ \wedge \ x = y \\ &= \mathit{false} \end{aligned}$$

Hence, the program is deadlock-free.

7. (a)
$$\begin{array}{ll} x = m \ \mathbf{co} \ x \geq m & \text{, given} \\ x \geq n \ \ \mathbf{co} \ \langle \exists \ m \ :: \ m \geq n \ \wedge \ x \geq m \rangle & \\ & \text{, elimination theorem} \\ x \geq n \ \ \mathbf{co} \ x \geq n & \text{, simplifying} \end{array}$$

conversely,

$$\begin{array}{ll} x \geq n \ \ \mathbf{co} \ x \geq n & \text{, given} \\ x = m \ \mathbf{co} \ x \geq m & \text{, strengthening lhs and renaming} \end{array}$$

(b) Applying the elimination theorem to the antecedent

$$x \geq y \ \mathbf{co} \ \langle \exists \ m, n \ :: \ m \geq n \ \wedge \ [x, y = m, n \ \vee \ (m > n \ \wedge \ x, y = m - 1, n)] \rangle$$

Simplifying and weakening the rhs

$$x \geq y \ \mathbf{co} \ \langle \exists \ m, n \ :: \ x \geq y \ \vee \ x \geq y \rangle$$

Simplifying the rhs

$$x \geq y \text{ stable}$$

(c) Applying the elimination theorem to the antecedent, where k is a free variable

$$x + y = k \ \mathbf{co} \ \langle \exists \ m, n \ :: \ m + n = k \ \wedge \ [x, y = m, n \ \vee \ x, y = m + 1, n - 1] \rangle$$

Weakening the rhs

$$x + y = k \ \mathbf{co} \ \langle \exists \ m, n \ :: \ x + y = k \rangle$$

Simplifying the rhs

$$x + y = k \ \mathbf{co} \ x + y = k$$

Hence, $x + y$ constant.

Alternative Proof: Employ the result of Exercise (8b).

$$\frac{x = m \ \mathbf{co} \ f.x = f.m}{f.x \quad \text{constant}}$$

(d)
$x = m$ **co** $x \geq m$, x is nondecreasing
$y = n$ **co** $y \leq n$, y is nonincreasing
$x, y = m, n$ **co** $x \geq m \ \wedge \ y \leq n$, conjunction of the above two
$x \geq y$ **co** $\langle \exists\ m, n \ :: \ m \geq n \ \wedge \ x \geq m \ \wedge \ y \leq n \rangle$
, elimination theorem
$x \geq y$ **co** $x \geq y$, simplifying rhs

(e) For a free variable S, of the same type as s, we are given

$s \ \subseteq \ S$ stable
$s = S$ **co** $s \ \subseteq \ S$, strengthening lhs

Then, using a free integer variable m and using $|s|$ to denote the size of s

$|s| \leq m$ **co** $\langle \exists\ S \ :: \ |s| \leq m \rangle$, weakening rhs
$|s| \leq m$ stable , simplifying

(f)
p **co** $\langle \exists\ m \ :: \ p[x \ := \ m] \ \wedge \ (p[x \ := \ m] \ \Rightarrow \ p) \rangle$
, elimination theorem
p **co** $\langle \exists\ m \ :: \ p \rangle$, weakening rhs
p **co** p , p does not name m

8. (a)
$f.x \ \sim \ n$
co {elimination theorem}
$\langle \exists\ m \ :: \ f.m \ \sim \ n \ \wedge \ f.x \ \sim \ f.m \rangle$
$\Rightarrow$ {$\sim$ is transitive}
$\langle \exists\ m \ :: \ f.x \ \sim \ n \rangle$
$\Rightarrow$ {simplifying}
$f.x \ \sim \ n$

(b) Use the result in Exercise 8a with "=" in the place of "$\sim$"; note that "=" is transitive. Therefore, we conclude, for any n,

or, $f.x = n$ stable
or, $f.x$ constant

(c)
$f.x = n$
co {elimination theorem}
$\langle \exists\ m \ :: \ f.m = n \ \wedge \ x \ \sim \ m \rangle$
$\Rightarrow$ {f preserves $\sim$}
$\langle \exists\ m \ :: \ f.m = n \ \wedge \ f.x \ \sim \ f.m \rangle$
$\Rightarrow$ {weakening}
$\langle \exists\ m \ :: \ f.x \ \sim \ n \rangle$
$\equiv$ {simplifying}
$f.x \ \sim \ n$

9. (a) $(p \equiv q)$ constant

   (b) The following property describes the change in sx.

   For all integers m, n
   $x, sx = m, n$ **co** $x, sx = m, n \ \vee \ (x \neq m \ \wedge \ sx = n + x)$
   Weaken the rhs of the above to get
   $x, sx = m, n$ **co** $x, sx = m, n \ \vee \ sx = n + x$
   Now, we show $sx \geq x$ stable.

   $sx \geq x$
   $\equiv$ {substitution axiom with $x \geq 0$ invariant}
   $sx \geq x \ \wedge \ x \geq 0$
   co {elimination theorem on the last **co**-property}
   $\langle \exists \ m, n \ : \ n \geq m \ \wedge \ m \geq 0 \ : \ x, sx = m, n \ \vee \ sx = n + x \rangle$
   $\Rightarrow$ {simplifying}
   $sx \geq x$

   (c) Define c as follows. Here m, n, r are free.

   var c : integer
   initially $c = 0$
   $x, y, c = m, n, r$ **co** $x, y, c = m, n, r \ \vee$
   $(x, y \neq m, n \ \wedge \ x > y \ \wedge \ c = r + 1) \ \vee \ (x \leq y \ \wedge \ c = r)$

   Next, assuming that $x \leq y$ invariant, we show that $c = r$ stable, for any r; therefore $c = 0$ stable. Combining with the initial condition, $c = 0$, the result follows.
   Weakening the rhs of the premise

   $x, y, c = m, n, r$ **co** $x, y, c = m, n, r \ \vee \ x > y \ \vee \ c = r$
   Using $x \leq y$ invariant in the rhs and weakening the rhs
   $x, y, c = m, n, r$ **co** $c = r$
   Taking disjunction over all m, n
   $c = r$ **co** $c = r$

   (d) Using a free variable m,

   $$x > 5 \ \wedge \ x = m \ \textbf{co} \ x \geq m$$

   A better formulation is

   $$x > 5 \ \wedge \ x \geq m \quad \text{stable}$$

   (e) We introduce an auxiliary predicate, b. Predicate b encodes the fact that $x > 5$ has been *true* in the past (or now).

initially $b \equiv x > 5$
$x > 5 \Rightarrow b$ invariant
$b \wedge x > 0$ stable

(f) For a vertex i, let S_i be the set of its neighbors. Let w_i denote that vertex i is black. Then the formalization is the same as in the case of mutual waiting in a knot (Section 3.5.7) and the same proof applies.

10. Let c_i denote the set of particles in the i^{th} chamber, $0 \leq i \leq N$; let a_j, b_j be the sets of particles in the chambers at or above j and at or below j, respectively. That is, for any j, $0 \leq j \leq N$,

$a_j = \langle \cup\ i\ :\ j \leq i \leq N\ :\ c_i \rangle$ and
$b_j = \langle \cup\ i\ :\ 0 \leq i \leq j\ :\ c_i \rangle$
Thus, the set of particles, V, in the vessel is given by
$V = \langle \cup\ i\ :\ 0 \leq i \leq N\ :\ c_i \rangle$
or $V = a_0$ or $V = b_N$
The law of particle movement within the vessel is given by, for any x and i, $0 \leq i \leq N$
$x \in c_i$ **co** $x \in a_i$
The constraint of impermeability can be written as
$x \notin b_i$ **co** $x \notin c_i$
Now, we have to show, for all x and j, $0 \leq j \leq N$
(10a) $x \in a_j$ stable
(10b) $x \notin b_j$ stable
(10c) $x \in V$ constant

Proof of (10a), $x \in a_j$ stable:

From the law of particle movement
$x \in c_i$ **co** $x \in a_i$
Taking disjunction over all i, in the range $j \leq i \leq N$, for some j,
$\langle \exists\ i\ :\ j \leq i \leq N\ :\ x \in c_i \rangle$ **co** $\langle \exists\ i\ :\ j \leq i \leq N\ :\ x \in a_i \rangle$
Rewriting
$x \in \langle \cup\ i\ :\ j \leq i \leq N\ :\ c_i \rangle$ **co**
$x \in \langle \cup\ i\ :\ j \leq i \leq N\ :\ a_i \rangle$
Using the definition of a_j
$x \in a_j$ **co** $x \in a_j$

Proof of (10b), $x \notin b_j$ stable:

From the constraint of impermeability
$x \notin b_i$ **co** $x \notin c_i$
Taking conjunction over all i, $0 \leq i \leq j$,
$\langle \forall\ i\ :\ 0 \leq i \leq j\ :\ x \notin b_i \rangle$ **co** $\langle \forall\ i\ :\ 0 \leq i \leq j\ :\ x \notin c_i \rangle$
Rewriting
$x \notin \langle \cup\ i\ :\ 0 \leq i \leq j\ :\ b_i \rangle$ **co** $x \notin \langle \cup\ i\ :\ 0 \leq i \leq j\ :\ c_i \rangle$
Using the definition of b_j
$x \notin b_j$ **co** $x \notin b_j$

Proof of (10c), $x \in V$ constant:

From 10a, using 0 for j and that $V = a_0$,
$x \in V$ stable
From 10b, using N for j and that $V = b_N$,
$x \notin V$ stable
Combining the above two
$x \in V$ constant

11. First, express that p is a precondition for any change in x, using free variable m.

$$x = m\ \wedge\ \neg p\ \textbf{co}\ x = m$$

Next express that q is established as a postcondition whenever x is changed.

$$x = m\ \textbf{co}\ x = m\ \vee\ q$$

(a) Given $x\ :\ p\ \rightarrow\ q$ we have,

$$\begin{array}{lll} x = m\ \wedge\ \neg p & \textbf{co}\ x = m & \text{and,} \\ x = m & \textbf{co}\ x = m\ \vee\ q & \end{array}$$

Strengthen the lhs of the first property by $\neg p'$ and weaken the rhs of the second property by q' to obtain

$$\begin{array}{lll} x = m\ \wedge\ \neg p\ \wedge\ \neg p' & \textbf{co}\ x = m & \text{and,} \\ x = m & \textbf{co}\ x = m\ \vee\ q\ \vee\ q' & \end{array}$$

i.e., $x\ :\ p\ \vee\ p'\ \rightarrow\ q\ \vee\ q'$

(b) We have,

(1) $x = m \ \wedge \ \neg p \ \ \mathbf{co} \ x = m$
(2) $x = m \qquad \mathbf{co} \ x = m \ \vee \ q$
(3) $x = m \ \wedge \ \neg p' \ \mathbf{co} \ x = m$
(4) $x = m \qquad \mathbf{co} \ x = m \ \vee \ q'$

Taking disjunction of (1,3)
$x = m \ \wedge \ (\neg p \ \vee \ \neg p') \ \mathbf{co} \ x = m$
Taking conjunction of (2,4)
$x = m \ \mathbf{co} \ x = m \ \vee \ (q \ \wedge \ q')$
Hence,
$x \ : \ p \ \wedge \ p' \ \rightarrow \ q \ \wedge \ q'$

12. Only the **assign**-sections of the programs are shown.

(a) $x \ := \ x$ {this is also known as the *skip* statement}
(b) $x \ := \ x + 1 \quad \text{if} \quad x = 0$
(c) $x \ := \ x + 1 \quad \text{if} \quad x > 0$
(d) x is a root of 64 iff $x = 2, 4, 8, -2,$ or -8. Hence,

$$x \ := \ x + 1 \qquad \text{if} \quad x \notin \ \{2, 4, 8, -2, -8\}$$

13. The problem is to show that

$$FP \ \equiv \ \langle \forall \ x \ \ :: \ \ x = 0 \rangle$$

From the description of the game, $FP \ \equiv \ \langle \forall \ x \ \ :: \ \ x = |x - x'| \rangle$. Therefore, we have to show that $\langle \forall \ x \ \ :: \ \ x = 0 \rangle = \langle \forall \ x \ \ :: \ \ x = |x - x'| \rangle$. Clearly, $\langle \forall \ x \ \ :: \ \ x = 0 \rangle \ \Rightarrow \ \langle \forall \ x \ \ :: \ \ x = |x - x'| \rangle$. We prove the converse, $\langle \forall \ x \ \ :: \ \ x = |x - x'| \rangle \ \Rightarrow \ \langle \forall \ x \ \ :: \ \ x = 0 \rangle$.

Assume $\langle \forall \ x \ \ :: \ \ x = |x - x'| \rangle$. For any x
$x = 0$
$\Rightarrow$ $\{x = |x - x'|\}$
$0 = |-x'|$
$\Rightarrow$ {property of absolute value function}
$x' = 0$
Applying induction, $\langle \exists \ x \ \ :: \ \ x = 0 \rangle \ \Rightarrow \ \langle \forall \ x \ \ :: \ \ x = 0 \rangle$.
Let y be the maximum value in the cycle.
true
$\Rightarrow$ $\{\langle \forall \ x \ \ :: \ \ x = |x - x'| \rangle\}$
$y = |y - y'|$
$\Rightarrow$ $\{y - y' \geq 0$ since y is the maximum value$\}$

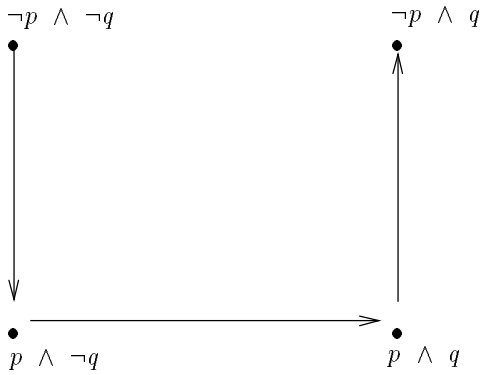


Figure 3.4: State Transition Diagram for Exercise 15

$$
\begin{array}{ll}
 & y = y - y' \\
\Rightarrow & \{\text{arithmetic}\} \\
 & y' = 0 \\
\Rightarrow & \{\text{previous proof}\} \\
 & \langle \forall\ x \ :: \ x = 0 \rangle
\end{array}
$$

14. First we show that FP is a solution to (14a). For arbitrary b

$$
\begin{array}{l}
\langle \forall\ p \ : \\
\quad \langle \forall\ q \ :: \ p \ \wedge \ q \ \text{ stable} \rangle \ : \ (p \ \wedge \ b \ \text{ stable}) \\
\rangle \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{, tautology} \\
\quad \langle \exists\ p \ : \ \langle \forall\ q \ :: \ p \ \wedge \ q \ \text{ stable} \rangle \ : \ p \ \wedge \ b \rangle \ \text{ stable} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{, disjunction of stables} \\
\quad \langle \exists\ p \ : \ \langle \forall\ q \ :: \ p \ \wedge \ q \ \text{ stable} \rangle \ : \ p \ \rangle \ \wedge \ b \ \text{ stable} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{, rewriting} \\
\quad FP \ \wedge \ b \ \text{ stable} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{, using the definition of } FP \\
\quad \langle \forall\ b \ :: \ FP \ \wedge \ b \ \text{ stable} \rangle \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{, binding the free variable } b
\end{array}
$$

Next, we show that FP is the weakest solution to (14a). Let x be any solution to 14a. Then

$\langle \forall\ b \ :: \ x \ \wedge \ b \text{ stable} \rangle$.

From the definition of FP, $x \ \Rightarrow \ FP$.

15. The state transition diagram is shown in Figure 4. The **co**-properties are obtained directly from the diagram.

$$
\begin{array}{ll}
\neg p \ \wedge \ \neg q \ \mathbf{co} \ \neg q & \{(\neg p \ \wedge \ \neg q) \ \vee \ (p \ \wedge \ \neg q) \ \equiv \ \neg q\} \\
\quad p \ \wedge \ \neg q \ \mathbf{co} \ p &
\end{array}
$$

$$p \ \wedge \ q \ \mathbf{co} \ q$$
$$\neg p \ \wedge \ q \text{ stable}$$

By taking suitable disjunctions we obtain

$$p \ \mathbf{co} \ p \ \vee \ q$$
$$\neg p \ \mathbf{co} \ \neg p \ \vee \ \neg q$$
$$q \ \mathbf{co} \ q$$
$$\neg q \ \mathbf{co} \ p \ \vee \ \neg q$$

To see that the second group of properties is equivalent to the first, take suitable conjunctions of the properties in the second group to obtain the first group.

16. For a ring network, we use the following notation. For any process i,

i' = the identity of the process to which i sends messages
$q.i$ = i is idle
$s.i$ = the number of messages sent by i
$r.i$ = the number of messages received by i

Analogous to CN1–CN4 we have, for all i, and free variables m, n

$$s.i \geq r.i' \geq 0 \qquad \text{(CN}'1\text{)}$$
$$s.i \geq m \text{ stable}, \quad r.i \geq n \text{ stable} \qquad \text{(CN}'2\text{)}$$
$$q.i \ \wedge \ r.i = m \ \mathbf{co} \ r.i = m \ \Rightarrow \ q.i \qquad \text{(CN}'3\text{)}$$
$$q.i \ \wedge \ s.i = n \ \mathbf{co} \ s.i = n \qquad \text{(CN}'4\text{)}$$

The property analogous to CN5 can be written using a set of free variables, $m.i$, one for each process i

$$\langle \forall \ i \ :: \ q.i \ \wedge \ r.i' = s.i \rangle \ \wedge \ \langle \forall \ i \ :: \ r.i = m.i \rangle \text{ stable} \qquad \text{(CN}'5\text{)}$$

Proof of CN′5: The proof is similar to the proof of (CN5). Conjoining (CN′3, CN′4)

$$q.i \ \wedge \ r.i, s.i = m, n \ \mathbf{co} \ (r.i = m \ \Rightarrow \ q.i) \ \wedge \ s.i = n$$

Replacing m by $m.i$ and n by $m.i'$

$$q.i \ \wedge \ r.i, s.i = m.i, m.i' \ \mathbf{co} \ (r.i = m.i \ \Rightarrow \ q.i) \ \wedge \ s.i = m.i'$$

Taking conjunction over all i

$$\langle \forall \ i \ :: \ q.i \ \wedge \ r.i, s.i = m.i, m.i' \rangle \ \mathbf{co}$$
$$\langle \forall \ i \ :: \ (r.i = m.i \ \Rightarrow \ q.i) \ \wedge \ s.i = m.i' \rangle$$

Since $r.i \geq m.i$ stable—from (CN′2)—its conjunction over all i is stable. Conjoining $\langle \forall \ i \ :: \ r.i \geq m.i \rangle$ to both sides

$$\langle \forall \ i \ :: \ q.i \ \wedge \ r.i, s.i = m.i, m.i' \rangle \ \mathbf{co}$$
$$\langle \forall \ i \ :: \ (r.i = m.i \ \Rightarrow \ q.i) \ \wedge \ s.i = m.i' \rangle \ \wedge$$
$$\langle \forall \ i \ :: \ r.i \geq m.i \rangle$$

In the rhs using $s.i = m.i', r.i' \geq m.i'$ and $s.i \geq r.i'$ (from CN′1) we get $r.i' = m.i'$. That is, $\langle \forall\ i \quad :: \quad r.i = m.i \rangle$. Simplifying the rhs, the result follows.

For the case of a general network, the treatment is entirely analogous, though a more general notation is needed. Let z denote the set of all nodes. From the suggested notation, iz denotes the set of outgoing channels from i and zi the set of incoming channels to i, and zz the set of all channels. For free variables m, n, a set of free variables, L and N, any channel c and process i, we are given

$$s.c \geq r.c \geq 0 \qquad \text{(CN″1)}$$
$$s.c \geq m \text{ stable}, \ r.c \geq n \text{ stable} \qquad \text{(CN″2)}$$
$$q.i \ \wedge \ (r = L).zi \ \mathbf{co} \ (r = L).zi \ \Rightarrow \ q.i \qquad \text{(CN″3)}$$
$$q.i \ \wedge \ (s = N).iz \ \mathbf{co} \ (s = N).iz \qquad \text{(CN″4)}$$

We have to show—where $q.z \ \equiv \ \langle \forall\ i \quad :: \quad q.i \rangle$—

$$q.z \ \wedge \ (s = r).zz \ \wedge \ (r = L).zz \text{ stable} \qquad \text{(CN″5)}$$

The proof is similar to the one for a ring. It is useful to note that $(s = r).xy$ and $(s \geq r).xy$ are both *true* if x of y is empty. Also,

$$(s = r).(x \cup x')(y \cup y') = (s = r).xy \ \wedge \ (s = r).xy' \ \wedge \ (s = r).x'y \ \wedge \ (s = r).x'y'$$

Similarly for $(s \geq r).(x \cup x')(y \cup y')$ □

17. We give an explicit construction for ss, similar to the construction for the strongest invariant (Section 3.6.2). Consider the infinite sequence of properties, $i \geq 0$,

$$q_i \ \mathbf{co} \ q_{i+1}$$

where $q_0 = p$ and q_{i+1} is the strongest rhs for q_i.

Let $ss.p = \langle \exists\ i \quad :: \quad q_i \rangle$.

$$p \ \Rightarrow \ ss.p$$

because $q_0 \ \Rightarrow \ \langle \exists\ i \quad :: \quad q_i \rangle$. Also, $ss.p$ is stable, according to the lemma in Section 3.6.2.

Finally, the proof that $ss.p$ is the strongest stable predicate weaker than p is along the same lines as for the proof of the strongest invariant. (In fact, $ss.p$ is the strongest invariant when p is the initial condition.)

(a) $(p \ \Rightarrow \ q) \ \Rightarrow \ (ss.p \ \Rightarrow \ ss.q)$

(a1) $p \;\Rightarrow\; ss.p \;\wedge\; ss.q$:

$p \;\Rightarrow\; q$	, antecedent
$q \;\Rightarrow\; ss.q$	, fact about ss
$p \;\Rightarrow\; ss.q$	, above two
$p \;\Rightarrow\; ss.p$	, property of ss
$p \;\Rightarrow\; ss.p \;\wedge\; ss.q$	, above two

(a2) $ss.p \;\wedge\; ss.q$ stable:

$ss.p$ stable	, fact about ss
$ss.q$ stable	, similarly
$ss.p \;\wedge\; ss.q$ stable	, conjunction of stable predicates

From $(a1, a2)$ $ss.p \;\wedge\; ss.q$ is a stable predicate and p implies this predicate. Since $ss.p$ is the strongest such predicate

$ss.p \;\Rightarrow\; (ss.p \;\wedge\; ss.q)$

i.e., $ss.p \;\Rightarrow\; ss.q$

(b) p stable $\equiv\; (ss.p = p)$

(b1) p stable $\Rightarrow\; (ss.p = p)$

Assume p stable. Since $p \;\Rightarrow\; p$ and p stable, and $ss.p$ is the strongest stable predicate that p implies, we get

$ss.p \;\Rightarrow\; p$

Also, $p \;\Rightarrow\; ss.p$, a fact about ss

Therefore, $p = ss.p$

(b2) $(ss.p = p) \;\Rightarrow\; p$ stable

Assume $ss.p = p$. Since $ss.p$ is stable, by definition, so is p.

(c) $ss.(ss.p) = ss.p$

For any p, $ss.p$ stable (a fact about ss). Substituting $ss.p$ for p in (17b1) above, we conclude

$$ss.(ss.p) = ss.p$$

(d) $ss.\langle \exists\; i \;::\; p_i \rangle = \langle \exists\; i \;::\; ss.p_i \rangle$

(d1) $ss.\langle \exists\; i \;::\; p_i \rangle \;\Rightarrow\; \langle \exists\; i \;::\; ss.p_i \rangle$:

From the definition of ss

$\langle \forall\; i \;::\; p_i \;\Rightarrow\; ss.p_i \rangle$

Taking disjunction over both sides of implication

$\langle \exists\; i \;::\; p_i \rangle \;\Rightarrow\; \langle \exists\; i \;::\; ss.p_i \rangle$

Also, from the definition ss

$\langle \forall\; i \;::\; ss.p_i$ stable$\rangle$

Since disjunctions of stable properties are stable

$\langle \exists\ i\ ::\ ss.p_i \rangle$ stable

Therefore, $\langle \exists\ i\ ::\ ss.p_i \rangle$ is a stable predicate weaker than $\langle \exists\ i\ ::\ p_i \rangle$. Since $ss.\langle \exists\ i\ ::\ p_i \rangle$ is the strongest such predicate, $ss.\langle \exists\ i\ ::\ p_i \rangle \Rightarrow \langle \exists\ i\ ::\ ss.p_i \rangle$

(d2) $\langle \exists\ i\ ::\ ss.p_i \rangle \Rightarrow ss.\langle \exists\ i\ ::\ p_i \rangle$

From predicate calculus,

$\langle \forall\ i\ ::\ p_i \Rightarrow \langle \exists\ i\ ::\ p_i \rangle\ \rangle$

From the monotonicity of ss

$\langle \forall\ i\ ::\ ss.p_i \Rightarrow ss.\langle \exists\ i\ ::\ p_i \rangle\ \rangle$

Taking disjunction over both sides of the above implication

$\langle \exists\ i\ ::\ ss.p_i \rangle \Rightarrow ss.\langle \exists\ i\ ::\ p_i \rangle$

Bibliography

[1] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 24(4):181–185, 1985.

[2] K. Mani Chandy and Jayadev Misra. How processes learn. *Journal of Distributed Computing*, 1:40–52, 1986.

[3] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison Wesley, 1988.

[4] K. Mani Chandy and Jayadev Misra. *Developments in Concurrency and Communication, ed. C. A. R. Hoare*, chapter Proofs of Distributed Algorithms: An Exercise. University of Texas at Austin Year of Programming. Addison-Wesley, 1990.

[5] Ernie Cohen. *Modular Progress Proofs of Concurrent Programs*. PhD thesis, The University of Texas at Austin, August 1992.

[6] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[7] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 1990.

[8] R. W. Floyd. Assigning meanings to programs. In T. Schwartz, editor, *Mathematical Aspects of Computer Science (Proc. Sym. in Applied Math)*, volume 19, pages 19–32. Amer. Math. Soc., 1967.

[9] J. Gray. Notes on database operating systems. In R. Bayer, R. M. Graham, and G. Seegmuller, editors, *Operating Systems: An Advanced Course*, volume 66 of *Lecture Notes in Computer Science*. Springer-Verlag, 1978. Also appears as *IBM Research Report RJ 2188*, August 1987.

[10] Joseph Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. Technical Report TR RJ 4421, IBM Almaden Research Center, 1989.

[11] C. A. R. Hoare. An axiomatic basis for computer programming. *C.ACM*, 12:476–580, 1969.

[12] C. S. Jutla, E. Knapp, and J. R. Rao. A predicate transformer approach to semantics of parallel programs. In *Proc. 8th ACM SIGACT/SIGOPS Symposium on Principles of Distributed Systems (PODC '89)*, pages 249–263, Edmonton, Alberta, Canada, 1989.

[13] E. Knapp. *Refinement as a Basis for Concurrent Program Design.* PhD thesis, The University of Texas at Austin, May 1992.

[14] Simon S. Lam and A. Udaya Shankar. Refinement and projection of relational specifications. In *Proc. REX Workshop on Stepwise Refinement of Distributed Systems, Plasmolen-Mook, The Netherlands.* LNCS Series, Springer-Verlag, May 1989.

[15] L. Lamport. Proving the correctness of multiprocess programs. *IEEE, Trans. on Software Engineering*, SE-3(2):125–143, March 1977.

[16] L. Lamport. *win* and *sin*: Predicate transformers for concurrency. *ACM Transactions on Programming Languages and Systems*, 12(3):396–428, 1990.

[17] L. Lamport. The temporal logic of actions. Technical Report SRC Research Report Number TR 79, Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave., Palo Alto, CA 94301, December 1991.

[18] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, New York, 1992.

[19] J. Misra. A theorem about dynamic acyclic graphs. Notes on UNITY: 02–88, September 1988.

[20] J. Misra. A foundation of parallel programming. In Manfred Broy, editor, *Proc. 9th International Summer School on Constructive Methods in Computer Science*, volume F 55 of *NATO ASI Series*, pages 397–433, Marktoberdorf, Germany, July, 1988, 1989. Springer-Verlag.

[21] J. Misra. Auxiliary variables. Notes on UNITY: 15–90, July 1990.

[22] J. Misra. Soundness of the substitution axiom. Notes on UNITY: 14–90, March 1990.

[23] J. R. Rao. *Building on the Unity Experience: Compositionality, Fairness and Probability in Parallelism.* PhD thesis, The University of Texas at Austin, August 1992.

[24] B. A. Sanders. Eliminating the substitution axiom from unity logic. *Formal Aspects of Computing*, 3:189–205, 1991.

[25] Mark G. Staskauskas. *Specification and Verification of Large-Scale Reactive Programs.* PhD thesis, The University of Texas at Austin, May 1992.

[26] Jan L. A. van de Snepscheut. For the record: Scheduling a meeting. Technical Report JAN–141, Groningen University, The Netherlands, January 1988.

[27] A. J. M. van Gasteren and G. Tel. On the proof of a distributed algorithm. *Information Processing Letters*, 35(6), 1990. Letter to the Editor.