Chapter 4
Progress

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

May 14, 1999

[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

4 Progress **5**

4.1 Introduction 5
4.2 Fairness 6
4.2.1 Minimal Progress 7
4.2.2 Weak Fairness 7
4.2.3 Strong Fairness 8
4.2.4 Which is the Fairest One? 8
4.3 Transient Predicate 9
4.3.1 Minimal Progress 10
4.3.2 Weak Fairness 11
4.3.3 Strong Fairness 12
4.3.4 Derived Rules 12
4.3.5 Discussion 13
4.4 ensures, leads-to 14
4.4.1 ensures 14
4.4.2 leads-to 14
4.4.3 Derived Rules 18
4.4.4 Proofs of the Derived Rules 20
4.4.5 Corollaries of the Derived Rules 25
4.5 Applications 26
4.5.1 Nonoperational Descriptions of Algorithms 27
4.5.2 Common Meeting Time 28
4.5.3 Token Ring 29
4.5.4 Unordered Channel 30

4.5.5 Dynamic Graphs 31
4.5.6 Strong Semaphore: An Example of Strong Fairness . 33
4.6 Theoretical Issues 35
4.6.1 *wlt* 35
4.6.2 A Fixpoint Characterization of *wlt* 37
4.6.3 The Role of the Disjunction Rule 38
4.7 Synopsis 40
4.8 Exercises 41
4.9 Solutions to Exercises 47

References **59**

4
Progress

4.1 Introduction

Safety properties, discussed in the last chapter, allow us to state that "the program does no harm." Such properties impose an "upper bound" on the set of allowable execution sequences of the program. A trivial program that causes no state change—a program consisting of a *skip* statement, for instance—satisfies all the safety properties. Thus, safety properties alone are insufficient as a basis of program design. Several formal aspects of program design and refinement are seriously affected by the absence of a "lower bound," that the program is required to have certain execution sequences.

In this chapter, we study a class of properties known as *progress*. Progress properties state that "the program does some good." For instance, "I press the switch and then the light is on" is an example of a progress property. A safety property for this system might be "the light never comes on unless the switch is pressed." This safety property is conveniently implemented by smashing the light bulb. Conversely, the given progress property might be implemented by having a light that is permanently on. It is the interplay between the safety and progress properties that determines a nontrivial design.

A progress property may be regarded as a performance guarantee. Such guarantees typically include time bounds: The light comes on within 10 ms of pressing the switch or a car travelling at 60 mph stops within 130 feet after the brakes are jammed. Absolute performance guarantees, though

desirable, are hard to implement, because such guarantees depend on the speed of the underlying machine or the network, the scheduling strategy or even the load on the system, factors that are outside our control during program design. A useful abstraction employed in complexity theory is to specify the rate of growth of the computation time as a function of the input size. This abstraction, effectively, ignores speed-ups by constant factors. An even coarser abstraction is to classify the rate of growth as being polynomial or non-polynomial. Unfortunately, we don't yet have a theory to provide such performance guarantees for the asynchronous systems that we consider here; we don't even have the appropriate parameters by which to measure a problem size. So, we abstract further by eliminating the notion of absolute time. We will state and prove properties of the form: Once predicate p holds, eventually q will hold in the system. For the lighting problem, p might be "the switch has been pressed" and q might be "the light is on." The time duration between the occurrences of p and q is left unspecified. We will develop the logic to state and deduce such properties. Realize, however, that this is a far cry from the absolute performance guarantees that we sought. However, it is a useful first step in establishing the performance of the system. Once we have such a guarantee we may attempt to deduce the performance empirically or by using analytic modelling.

Chapter Outline

In Section 4.2, we describe several notions of fairness that are essential for studying progress properties in asynchronous systems. In Section 4.3, we introduce *transient* predicates; a transient predicate is guaranteed to be falsified eventually, under the given fairness assumption. Our primary progress operator, *leads-to*, is introduced in Section 4.4 . Most progress specifications and deductions are done with *leads-to*. So, we give a variety of manipulation rules and show several examples of *leads-to* in Section 4.5. Certain theoretical issues are taken up in Section 4.6. A synopsis appears in Section 4.7.

4.2 Fairness

The need for fairness and the various flavors of it can be explained by considering the "guarded-command" program shown below. The guards for the statements α, β, γ are $true, true, x \neq y$, respectively. In this program, x, y, z are integers.

Program Fairness

$$
\begin{array}{lcl}
\alpha & :: & x := x+1 \\
[] \ \beta & :: & y := y+1 \\
[] \ \gamma & :: & z := z+1 \qquad \text{if} \quad x \neq y
\end{array}
$$

end {Fairness}

A fairness condition defines the order in which the statements, α, β, γ are executed. We study the following kinds of fairness: minimal progress, weak fairness and strong fairness.

4.2.1 Minimal Progress

Under this notion of fairness, the following action is repeated until all guards are false; a non*skip* statement whose guard is true in the current state is chosen (arbitrarily) and executed.

This fairness condition guarantees that if it is possible for the system to change state, it will do so. For the above example, minimal progress forbids executing γ forever starting in a state where $x = y$, thus preserving the same state. We can assert for this program that $x + y + z$ will eventually increase (and hence, it will increase without bound). This is because all guards are never *false*—α, β have *true* as their guards—and any change in the system state increases $x + y + z$. However, neither x, y nor z is guaranteed to increase because, for instance, β (or α or γ) might be executed indefinitely. Similarly, no guarantee can be made that $x + y$ will increase (γ might be executed forever once $x \neq y$); there is no guarantee about eventual increase of $x + z$ or $y + z$ either.

4.2.2 Weak Fairness

Under this notion of fairness, in an infinite execution each statement is executed infinitely often.

Executing a statement in a state where its guard is *false* causes no state change. This fairness condition guarantees that different processes in a multiprocess program will be individually allowed to proceed. We can imagine that the statements representing the various processes constitute the program under consideration. For example, α might belong to one process and β, γ to another. If α is chosen forever for execution (as in minimal progress), we have, effectively, permanently blocked the second process; weak fairness prevents such executions.

For the example program we can assert that x (and y) will increase without bound because each execution of α (or β) will cause x (or y) to increase. We cannot assert that z will increase. For instance, consider the following execution starting in state $x, y = 0, 0$: Execute α, β, γ in this order and repeat the sequence forever. Whenever γ is executed $x = y$ and hence, z is never increased.

Weak fairness is sometimes expressed as: If the guard of a statement remains continuously *true* then the statement is eventually executed (in a state where the guard is *true*). This formulation is identical to our earlier formulation.

4.2.3 Strong Fairness

Execution of a statement is strongly fair if in every execution sequence, whenever the guard of the statement is infinitely often *true*, the statement is executed (in a state where the guard is *true*) infinitely often.

For the above example, if all three statements are executed in a strongly fair manner then x, y, z will all increase indefinitely. It is easy to see this result for x, y; for z, note that $x \neq y$ is infinitely often *true* since x, y are incremented asynchronously (see Exercise 11); therefore, z will be incremented infinitely often.

4.2.4 Which is the Fairest One?

We may mix different kinds of fairness for the same program; we may group the statements and require different forms of fairness for each group. For instance, if we require minimal progress for $\{\alpha, \beta\}$ and strong fairness for $\{\gamma\}$ in the program **Fairness**, then we can assert that $x + y$ and z will increase indefinitely, but neither x nor y can be asserted to increase. If $\{\alpha, \beta\}$ have the weak fairness restriction and $\{\gamma\}$ the strong fairness then all three—x, y, z—will grow indefinitely.

Grouping statements introduces the possibility of hierarchy; subgroups within a group may have different fairness conditions attached to them. Though these possibilities are interesting theoretically, we do not pursue them in this book. In fact, we develop the theory only for minimal progress and weak fairness and, partially, for strong fairness.

A traditional sequential program permits no choice in statement executions. Therefore, the only pertinent notion of fairness in this case is minimal progress. The significant progress property is termination; it can be stated as "starting in any state that satisfies the initial condition eventually a state is reached that satisfies FP," (FP is the fixedpoint predicate; a state satisfying FP is a terminal state). Termination can be proved by displaying a function whose value decreases eventually as a result of program execution; if there is a lower bound on the function value (specifically, if the function assumes values from a well-founded set) its value cannot decrease forever, and hence, termination is guaranteed. Minimal progress is also useful in concurrent programs for proving the "absence of deadlock"; a typical example is if there is a hungry philosopher (in a dining philosophers problem) then some philosopher will eat.

Minimal progress is not sufficient to guarantee "absence of individual starvation." Even though some philosopher may eat, and eating is performed infinitely often, a particular philosopher may stay hungry forever (and starve). In the program **Fairness** studied in this section, the system as a whole makes progress by increasing $x + y + z$; no guarantees could be made about the individual variables—x, y or z—though.

Weak fairness meets most of our criteria for a useful notion of fairness.

It is powerful enough so that starvation-free solutions can be designed, and it is simple enough that a reasonably effective theory for reasoning about it can be developed. It has nice compositional properties (see Chapter 5).

A typical example of the application of strong fairness is in implementing a "strong" semaphore. It is required that if the semaphore value exceeds 0 infinitely often then every process waiting for the semaphore be granted the semaphore. Under the weak fairness requirement a waiting process may never be granted the semaphore (because the guard—that the semaphore value exceeds 0—is not continuously *true*, but it is *true* infinitely often.) We take up this example in Section 4.5.5.

4.3 Transient Predicate

A predicate is *transient* if it is guaranteed to be falsified by execution of a single (atomic) transition. The formal definition depends on the form of fairness assumed for program execution. This is the only point in our theory where a definition of an operator depends on the form of fairness. Other progress operators are defined using transient predicates; their definitions and derived rules are independent of the underlying fairness. Thus, our progress proofs are largely shielded from having to argue about specific fairness properties of programs. We consider different notions of fairness and define transient predicates for each case.

There is a variety of language features for process synchronization, mutual exclusion, process communication, etc. It is possible, though cumbersome, to define transient predicates for programs that include these programming language features. For simplicity, we will restrict ourselves to transition systems; such systems can be represented by a UNITY program[3]. We make two assumptions about such systems.

- There is at least one transition. Recall that *skip* is included as an action in every system and, hence, this requirement is always met. (We do not show *skip*, explicitly, in the programs we write.)
- Each transition terminates, i.e., if the transition is started in any state where it is enabled, it completes in finite time. It is easy to check termination for simple transitions, such as the ones that can be represented by assignment of values to variables. For more intricate transitions, for instance, where a transition is an entire program, the methods of this chapter have to be applied recursively to first prove terminations of the individual transitions and then deduce progress properties of the system.

For a terminating transition s, we have the Law of Excluded Miracle[7]

$$\frac{\{p\}\ \ s\ \ \{false\}}{\neg p}$$

i.e., the postcondition of a transition is *false* only if the precondition is *false*. Using the substitution axiom, this can be interpreted as "the resulting state of a transition is unreachable only if the transition is started in an unreachable state."

4.3.1 Minimal Progress

We consider a program of guarded commands, without *skip* commands, of the form

$$\langle [\!] \; i \;\; :: \;\; g_i \;\Rightarrow\; s_i \rangle$$
or, $\langle [\!] \; i \;\; :: \;\; s_i \;\; \text{if} \;\; g_i \rangle$ (in UNITY)

Predicate p is **transient** if,

(1) whenever p holds some command has a *true* guard:
$$p \;\Rightarrow\; \langle \exists \; i \;\; :: \;\; g_i \rangle$$

and, (2) executing *any* command that has a *true* guard in a state where p holds falsifies p:
$$\langle \forall \; i \;\; :: \;\; \{p \wedge g_i\} \;\; s_i \;\; \{\neg p\} \rangle$$

Thus, a transient predicate, if *true* before a step, is *false* after the step. This definition may seem overly restrictive; can we not require that a predicate be falsified in a finite number of steps? The operator *leads-to* will be used to express such facts. Also, our requirement (2), that every enabled non*skip* transition falsify p is essential; without such a requirement a possible execution may consist only of transitions that never falsify p.

Example:

We consider the running example from Section 4.2, which we reproduce below.

Program Fairness
$$\begin{array}{lll} & \alpha :: x := x+1 & \\ [\!] & \beta :: y := y+1 & \\ [\!] & \gamma :: z := z+1 & \text{if } x \neq y \end{array}$$
end {Fairness}

First, we establish that for any integer k,

$$x + y + z = k \text{ transient}$$

Following are the proof obligations and they are easily established. For any integer k,

1. $x + y + z = k \;\Rightarrow\; true$

2. $\{x+y+z=k\}\ \ x\ :=\ x+1\ \ \{x+y+z\neq k\}$
   $\{x+y+z=k\}\ \ y\ :=\ y+1\ \ \{x+y+z\neq k\}$
   $\{x+y+z=k\ \wedge\ x\neq y\}\ \ z\ :=\ z+1\ \ \{x+y+z\neq k\}$

We can also show that $x=y$ transient. The proof obligations are similarly obtained. In this case, corresponding to statement γ we are required to show

$$\{x=y\ \wedge\ x\neq y\}\ \ z\ :=\ z+1\ \ \{x\neq y\}$$

which follows from $\{false\}\ \ s\ \ \{q\}$ for any statement s and predicate q.

Next we attempt proving for any k, $x=k$ transient. We know from operational arguments that this property does not hold. The proof obligations are

1. $x=k\ \Rightarrow\ true$

2. $\{x=k\}\ x\ :=\ x+1\ \ \{x\neq k\}$
   $\{x=k\}\ \ y\ :=\ y+1\ \ \{x\neq k\}$
   $\{x=k\ \wedge\ x\neq y\}\ \ z\ :=\ z+1\ \ \{x\neq k\}$

   The last two assertions cannot be established.

Finally, we leave it to the reader to show that neither $x+y=k$ nor $x+z=k$ can be shown transient. □

4.3.2 Weak Fairness

A transient predicate is falsified by *every* "enabled" transition under minimal progress. Under weak fairness, however, it is sufficient to have a single transition falsify the predicate. Define,

$$p \text{ transient } \underline{\Delta}\ \langle \exists\ s\ \ ::\ \ \{p\}\ \ s\ \ \{\neg p\}\rangle$$

where s is over all transitions in the system.

The following operational argument shows that eventually $\neg p$ holds given that p is transient. Let t be a transition that falsifies p. From the weak fairness condition, t is executed eventually. If $\neg p$ holds prior to the execution of t then the proof is over. Otherwise, p holds prior to the execution of t, and, from $\{p\}\ \ t\ \ \{\neg p\}$, following the execution of t, $\neg p$ holds. (Note that the execution of t is assumed to terminate.)

Example:

We consider the program **Fairness** of Section 4.2. The following predicates can be shown to be transient. For any integer k,

$$x=k\ ,\ y=k\ ,\ x+y=k\ ,\ y+z=k\ ,\ x+z=k\ ,\ x+y+z=k$$

The predicate $z = k$ cannot be shown transient, because we cannot display an appropriate statement. The only statement that modifies z is

$$z \ := \ z + 1 \qquad \text{if} \quad x = y$$

and, this statement does not satisfy

$$\{z = k\} \ \ z \ := \ z + 1 \qquad \text{if} \quad x = y \ \ \{z \neq k\}$$

The reader may also show that $x \leq k$ cannot be proven to be transient. □

4.3.3 Strong Fairness

Transient predicates for strong fairness can be defined using a recursive definition employing *leads-to*; see [11] for details (in this reference the progress operator *ensures* is defined for strong fairness; transient predicates and *ensures* are related–see Section 4.4.1). We do not plan to employ this definition nor consider strong fairness in any detail in this book. We give a rule in Section 4.5.6–that is sound but incomplete–to prove progress properties under strong fairness.

4.3.4 Derived Rules

We establish two simple rules about transient predicates that hold under either notion of fairness—minimal progress or weak fairness. These rules are primarily used in proving the derived rules for *leads-to*; these are rarely used in deducing properties of specific programs.

- The only predicate that is both stable and transient is *false*:

$$(p \text{ stable } \wedge \ p \text{ transient}) \ \equiv \ \neg p$$

- (strengthening)
$$\frac{p \text{ transient}}{p \ \wedge \ q \text{ transient}}$$

Additionally, substitution axiom can be used as usual.

It is easy to see from the definition of transient predicates that *false* is transient; it is known that *false* is stable. Therefore, in order to establish the first rule, we show, below, that

$$(p \text{ stable } \wedge \ p \text{ transient}) \ \Rightarrow \ \neg p$$

Proof of $(p \text{ stable} \wedge p \text{ transient}) \Rightarrow \neg p$ (under minimal progress)

For any statement with guard g_i and body s_i:

$\{p \wedge g_i\}\ s_i\ \{p\}$	, p stable
$\{p \wedge g_i\}\ s_i\ \{\neg p\}$	, p transient
$\{p \wedge g_i\}\ s_i\ \{false\}$	, conjunction of the above two
$\neg(p \wedge g_i)$	, from the Law of Excluded Miracle
$p \Rightarrow \neg g_i$	, simplifying the above
$p \Rightarrow \langle \forall\ i\ ::\ \neg g_i \rangle$	, conjoining over all i
$p \Rightarrow \langle \exists\ i\ ::\ g_i \rangle$	, definition of p transient
$\neg p$	, conjoining the above two □

Proof of (*p* stable) $\wedge$ (*p* transient) $\Rightarrow\ \neg p$ (under weak fairness)

From the definition of *p* transient, there is a statement *t* such that

$\{p\}\ t\ \{\neg p\}$	, p transient
$\{p\}\ t\ \{p\}$	, p stable
$\{p\}\ t\ \{false\}$	, conjunction of the above two
$\neg p$	, the Law of Excluded Miracle □

Proof of the strengthening rule (under minimal progress)

$p \Rightarrow \langle \exists\ i\ ::\ g_i \rangle$	, p transient
$p \wedge q \Rightarrow \langle \exists\ i\ ::\ g_i \rangle$	, predicate calculus

Also, for a statement with guard g_i and body s_i:

$\{p \wedge g_i\}\ s_i\ \{\neg p\}$	, p transient
$\{p \wedge q \wedge g_i\}\ s_i\ \{\neg p \vee \neg q\}$	, strengthening the lhs and weakening the rhs

Hence, $p \wedge q$ transient □

Proof of the strengthening rule (under weak fairness)

There is a statement *t* such that

$\{p\}\ t\ \{\neg p\}$	, p transient
$\{p \wedge q\}\ t\ \{\neg p \vee \neg q\}$	, strengthening the lhs and weakening the rhs
$p \wedge q$ transient	, definition of transient □

4.3.5 Discussion

The notion of stability (and its generalization in **co**) was fundamental in developing our theory of safety; a stable predicate is guaranteed never to

be falsified. The notion of transient, fundamental to a theory of progress, is almost the opposite of stability; a transient predicate is guaranteed to be falsified. A predicate can be established stable only by examining *all* transitions in a system (i.e., a universal quantification appears in its definition). Under weak fairness, a predicate can be established transient by examining *some* transition in a system (i.e., the definition involves existential quantification).

It is an interesting research question to identify problem areas where stability and transience are the only useful logical notions. For such cases, special purpose theories may be efficient for deriving system properties.

4.4 ensures, leads-to

Our primary progress operator is *leads-to*. It is the transitive, disjunctive closure of an operator, *ensures*. It is possible to eliminate *ensures* from our theory, replacing it by **co**-properties and transient predicates. However, to maintain continuity with [3], we introduce *ensures* though its role is now considerably diminished.

4.4.1 ensures

The informal meaning of p *ensures* q (abbreviated p **en** q) is: If p holds at any point in the computation it will continue to hold as long as q does not hold; eventually q holds; further, there is one (atomic) transition which guarantees to establish q starting in any p-state. Formally,

$$p \textbf{ en } q \triangleq (p \wedge \neg q \textbf{ co } p \vee q) \wedge (p \wedge \neg q \text{ transient})$$

It follows from the **co**-property in the above definition that once p holds it continues to hold as long as q does not. Now, we justify in operational terms that once p holds, q holds eventually ("eventually" includes the present moment). Consider a state in which p holds and q does not. Since $p \wedge \neg q$ is transient, it is eventually falsified. From the **co**-property, whenever $(p \wedge \neg q)$ is falsified, $(p \vee q)$ holds. Hence, whenever $(p \wedge \neg q)$ is falsified, $\neg(p \wedge \neg q) \wedge (p \vee q)$, i.e., q, holds.

4.4.2 leads-to

The informal meaning of $p \mapsto q$ (read: p *leads-to* q) is "if p holds at any point in the computation then q will hold" (here 'will' applies to the current point as well as the future). There is no guarantee, unlike *ensures*, that p remains *true* until q holds.

The definition of $p \mapsto q$, given below, is recursive. For the basis, we deduce $p \mapsto q$ from p **en** q. The transitivity rule ($p \mapsto r$ can be deduced

from $p \mapsto q$ and $q \mapsto r$) is justified as follows. From $p \mapsto q$, once p holds q will hold, and from $q \mapsto r$, once q holds r will hold. Therefore, once p holds r will hold. The disjunctivity rule (given $p \mapsto q$, for all p in a certain set, deduce $P \mapsto q$ where P is the disjunction of the predicates in that set) is justified, because any state that satisfies P also satisfies some predicate, p, in the given set and hence, starting from this state q will eventually be established, from $p \mapsto q$. An alternative definition of *leads-to* that eliminates **en** appears in Exercise 4. Exercise 19 shows that the transitivity and the disjunction rules may be combined into a single rule.

(basis)
$$\frac{p \ \mathbf{en} \ q}{p \mapsto q}$$

(transitivity)
$$\frac{p \mapsto q \ , \ q \mapsto r}{p \mapsto r}$$

(disjunction) In the following, S is any set of predicates.

$$\frac{\langle \forall \ p \ : \ p \in S \ : \ p \mapsto q \rangle}{\langle \exists \ p \ : \ p \in S \ : \ p \rangle \mapsto q}$$

As usual, the substitution axiom applies.

The form of this definition—using inference rules—differs from the way we defined **co**, transient and **en**. The current definition is recursive, and $\mapsto$ can be understood as an extreme solution (a least fixpoint) of an equation, a topic that we explore in Section 4.6.2. The inference rules provide important guidelines for structuring progress proofs: Either a proof follows directly from the program text (in the basis case), or it has to be structured as a transitive or a disjunctive proof. These rules can also be used to establish derived rules for $\mapsto$ using structural induction over its definition (see Sections 4.4.4, 4.6.1).

Note:

The only mention of **en** is in the basis rule. The premise of that rule could be replaced by the definition of p **en** q—$p \wedge \neg q$ **co** $p \vee q$, $p \wedge \neg q$ transient—thus eliminating **en** from our theory. □

A Note on Quantification:

The range of quantification involving $\mapsto$ (or even **en**) can be manipulated as follows.

$$\langle \forall \ k \ : \ r \ : \ p \mapsto q \rangle \ \equiv \ \langle \forall \ k \ :: \ p \wedge r \mapsto q \rangle$$ □

Examples:

In the following, variables x, y are integers and S, T are finite sets of integers.

1. A hungry philosopher eats. Let h, e denote, respectively, that a particular philosopher is hungry or eating.

   $$h \;\mapsto\; e$$

2. Variable x changes eventually. For every integer m,

   $$x = m \;\mapsto\; x \neq m$$

   This can be written equivalently as (see Exercise 7d)

   $$\mathit{true} \;\mapsto\; x \neq m$$

3. Variable x grows without bound. For every integer m

   $$\mathit{true} \;\mapsto\; x > m$$

   Note that the above property is an abbreviation for

   $$\langle \forall\, m \quad :: \quad \mathit{true} \;\mapsto\; x > m \rangle \;.$$

   It should not be confused with

   $$\mathit{true} \;\mapsto\; \langle \forall\, m \quad :: \quad x > m \rangle$$

   which happens to be nonsense.

4. Every integer is added to S eventually. For every integer m,

   $$\mathit{true} \;\mapsto\; m \in S$$

   It is not possible to conclude from the above that S grows eventually, because there is no restriction on items being removed from S. We impose such a restriction: For any fixed set of integers U,

   $$S \;\supseteq\; U \quad \text{stable}$$

   It can now be shown that S grows without bound. However, it does not follow from the above two properties that S will eventually include all integers. A possible scenario is that initially S is empty and number m is added to S at the m^{th} step, $m \geq 0$, (and no number is ever removed from S). Then, S becomes arbitrarily large though it remains finite after a finite number of steps.

5. Variables x, y do not retain their values forever if they differ. For all integers m, n

   $$x, y = m, n \ \wedge \ m \neq n \ \mapsto \ x, y \neq m, n$$

   or, equivalently, $\langle \forall \ m, n \ : \ m \neq n \ : \ x, y = m, n \ \mapsto \ x, y \neq m, n \rangle$.

   Observe that there is no guarantee for x, y to ever become equal.

6. Every element common to S, T is eventually removed from both sets.

   $$m \ \in \ (S \ \cap \ T) \ \mapsto \ m \notin \ (S \ \cup \ T)$$

   It cannot be deduced from the above that S, T will eventually become disjoint.

7. Predicate p holds infinitely often.

   $$\mathit{true} \ \mapsto \ p$$

   or, equivalently (see Exercise 7d)

   $$\neg p \ \mapsto \ p$$

8. If p remains *true* forever then q holds eventually.

   Another way of expressing this property is to say that eventually either p is *false* or q holds:

   $$\mathit{true} \ \mapsto \ \neg p \ \vee \ q$$

9. If p holds infinitely often (in all executions) then so does q.

   $$(\mathit{true} \ \mapsto \ p) \ \Rightarrow \ (\mathit{true} \ \mapsto \ q)$$

   This property does *not* say that "In any execution, if p holds infinitely often then so does q." This latter property is stronger than our formulation. This is because, if in some executions p holds infinitely often and in some other executions p holds finitely often, the first formulation makes no guarantees about q in any execution; the second formulation requires q to hold infinitely often whenever p does.

10. A given program "terminates," i.e., starting in any state that satisfies the initial condition eventually a state is reached that satisfies the fixed point predicate, FP.

    $$\text{initial-condition} \ \mapsto \ FP$$ □

In Section 4.6.3, we show that the disjunction rule over a finite set, though useful in practice, can be deduced from the basis and transitivity rules. Therefore, the real power of the disjunction rule lies in its application to an infinite set of predicates.

4.4.3 Derived Rules

Effective applications of the following derived rules can shorten progress proofs by an order of magnitude (I speak from personal experience). The rules are divided into two classes, *lightweight* and *heavyweight*. The former includes rules whose validity is easily established; the latter set of rules are not entirely obvious. Each application of a heavyweight rule goes a long way toward completing a progress proof.

Lightweight Rules

- (implication) $$\frac{p \;\Rightarrow\; q}{p \;\mapsto\; q}$$

The remaining lightweight rules have counterparts for **co**.

- (lhs-strengthening, rhs-weakening) $$\frac{p \;\mapsto\; q}{\begin{array}{c} p' \;\wedge\; p \;\mapsto\; q\ , \\ p \;\mapsto\; q \;\vee\; q' \end{array}}$$

- (disjunction) In the following, i is quantified over any arbitrary set, and p_i, q_i are predicates.

$$\frac{\langle \forall\ i \;::\; p_i \;\mapsto\; q_i \rangle}{\langle \exists\ i \;::\; p_i \rangle \;\mapsto\; \langle \exists\ i \;::\; q_i \rangle}$$

- (cancellation) $$\frac{p \;\mapsto\; q \;\vee\; r\ ,\ r \;\mapsto\; s}{p \;\mapsto\; q \;\vee\; s}$$

We deduce from the implication rule that for any predicate p,

$$p \;\mapsto\; p \quad \text{and} \quad \mathit{false} \;\mapsto\; p\ .$$

The lhs-strengthening and the rhs-weakening rules—also valid for **co**-properties—are used extensively in proofs. The disjunction rule, given previously, is slightly more general than the one given here (for an empty set of predicates, the previous rule yields $\mathit{false} \;\mapsto\; q$ for any q, and the current rule yields $\mathit{false} \;\mapsto\; \mathit{false}$). We will not distinguish the two rules by name; it should be obvious in any application which rule is being considered. The cancellation rule played little role in manipulating the **co**-properties; it is, however, used heavily in progress proofs. The cancellation rule reduces to transitivity when q is *false*. Note that there is no conjunction rule for $\mapsto$ analogous to the one for **co** (see Exercise 12c).

Heavyweight Rules

- (impossibility) $$\frac{p \;\mapsto\; \mathit{false}}{\neg p}$$

- (PSP)
$$\frac{p \mapsto q\ ,\quad r\ \mathbf{co}\ s}{p \wedge s \mapsto (q \wedge r) \vee (\neg r \wedge s)}$$

- (induction) Let M be a total function from program states to the set W. Let $(W, <)$ be well-founded. The variable m in the following premise ranges over W.

$$\frac{\langle \forall\ m \quad :: \quad p \wedge M = m \mapsto (p \wedge M < m) \vee q \rangle}{p \mapsto q}$$

where p, q do not name m.

- (completion)

Let i take on a finite set of values, and let p_i, q_i be predicates for each i.

$$\frac{\langle \forall\ i \quad :: \quad p_i \mapsto q_i \vee b \quad ,\quad q_i\ \mathbf{co}\ q_i \vee b \rangle}{\langle \forall\ i \quad :: \quad p_i \rangle \mapsto \langle \forall\ i \quad :: \quad q_i \rangle \vee b}$$

The impossibility rule says that a state satisfying *false* is reachable only from an unreachable state (read the consequent of the rule, "$\neg p$ invariant").

The PSP rule (for Progress-Safety-Progress) is perhaps the most widely used rule in progress proofs. It allows us to structure a progress proof as a safety proof—establishing r **co** s—and a progress proof—establishing $p \mapsto q$—which are then combined. This rule is so important that it should be memorized before attempting serious progress proofs.

Function M in the induction rule is called a *variant function* or a *metric*. The premise of the induction rule says that from any state in which p holds, eventually a state is reached where p still holds and the metric has a lower value, or q is established. Since M takes values from a well-founded set, its value cannot decrease indefinitely. Therefore, q is eventually established starting in any state where p holds. (Note: It is sufficient to require that M's value be in W whenever $p \wedge \neg q$ holds.) Some common examples of well-founded relations are: less-than relation over positive integers (or natural numbers), lexicographic order over tuples (where the corresponding items in all tuples are totally ordered), proper prefix or proper subsequence relation over finite sequences and proper subset relation over finite sets.

The completion rule is a way to take conjunctions of progress properties. As we have remarked earlier, there is no conjunction rule for $\mapsto$, analogous to the rule for **co**-properties (Exercise 12c). Under additional assumptions about the predicates in the rhs of the *leads-to*'s (given by the **co**-properties),

such a conjunction rule is valid. Exercise 16 asks you to show that the rule is *not* valid for an infinite pair of predicates, p_i and q_i; certain generalizations of this rule appear in Exercise 17.

4.4.4 Proofs of the Derived Rules

The lightweight rules can be proven directly from the infererence rules for $\mapsto$. The heavyweight rules often require induction on the structure of the progress proofs in the premises.

Proofs of the Lightweight Rules

- (implication) $$\frac{p \;\Rightarrow\; q}{p \;\mapsto\; q}$$

Proof:

$p \;\wedge\; \neg q \;\equiv\; \mathit{false}$	, from premise $p \;\Rightarrow\; q$
$p \;\wedge\; \neg q$ **co** $p \;\vee\; q$	, *false* **co** r, for any r
$p \;\wedge\; \neg q$ transient	, *false* transient
p **en** q	, definition of **en**
$p \;\mapsto\; q$	, from basis inference rule for $\mapsto$

- (lhs-strengthening; rhs weakening) $$\frac{p \;\mapsto\; q}{\begin{array}{l} p' \;\wedge\; p \;\mapsto\; q\;, \\ p \;\mapsto\; q \;\vee\; q' \end{array}}$$

Proof:

$p' \;\wedge\; p \;\mapsto\; p$	, implication
$p \;\mapsto\; q$	, premise
$p' \;\wedge\; p \;\mapsto\; q$	, transitivity on the above two

Similarly, $p \;\mapsto\; q \;\mapsto\; q \;\vee\; q'$.

- (disjunction) $$\frac{\langle \forall\; i \;::\; p_i \;\mapsto\; q_i \rangle}{\langle \exists\; i \;::\; p_i \rangle \;\mapsto\; \langle \exists\; i \;::\; q_i \rangle}$$

Proof:

If the range of quantification for i is empty, the conclusion, $\mathit{false} \;\mapsto\; \mathit{false}$, follows from implication. Assume, therefore, that range of i is nonempty.

$\langle \forall\; i \;::\; p_i \;\mapsto\; q_i \rangle$	, premise
$\langle \forall\; i \;::\; p_i \;\mapsto\; \langle \exists\; i \;::\; q_i \rangle\; \rangle$	, rhs-weakening

Applying the disjunction inference rule, the result follows.

- (cancellation) $$\frac{p \mapsto q \vee r \ , \ r \mapsto s}{p \mapsto q \vee s}$$

Proof:

$r \mapsto s$	, premise
$q \mapsto q$	, implication
$q \vee r \mapsto q \vee s$	, disjunction
$p \mapsto q \vee r$	, premise
$p \mapsto q \vee s$	, transitivity on the above two

Proofs of the Heavyweight Rules

In establishing these rules we sometimes employ "structural induction" on the proofs of the progress properties in the premise. Suppose, for instance, $p \mapsto q$ appears in the premise. This property could have been established either (1) from p **en** q or (2) from $p \mapsto r$, $r \mapsto q$ for some r, or (3) from a set of properties $r \mapsto q$, for each r in a set S, where $p = \langle \exists \ r \ : \ r \in S \ : \ r \rangle$. We have to prove the consequent in each of these three cases. For the first case, we draw upon the derived rules of **co** and transient (using which *ensures* is defined); for the remaining two cases, using the induction hypothesis, assume that the rule holds for each of the "smaller" progress properties and include these as premises. This is the strategy for the proofs of impossibility and PSP rules.

- (impossibility) $$\frac{p \mapsto \mathit{false}}{\neg p}$$

basis:

p **en** *false*	, premise
p stable and p transient	, definition of p **en** $false$
$\neg p$	, derived rule (see Section 4.3.4)

transitivity:

There is a predicate r such that $p \mapsto r \mapsto \mathit{false}$. Hence,

$\neg r$	, induction hypothesis
$p \mapsto \mathit{false}$	, from $p \mapsto r$ and $\neg r$
$\neg p$	, induction hypothesis

disjunction:

There is a set, S, of predicates such that $r \mapsto \mathit{false}$ for $r \in S$, and $p = \langle \exists\ r\ :\ r \in S\ :\ r \rangle$. Hence, for every r in S

$r \mapsto \mathit{false}$	, premise
$\neg r$	, induction hypothesis
$\neg p$	, from $p = \langle \exists\ r\ :\ r \in S\ :\ \mathit{false} \rangle$ $\quad \Box$

- (PSP) $\dfrac{p \mapsto q,\ r\ \mathbf{co}\ s}{p \wedge s \mapsto (q \wedge r) \vee (\neg r \wedge s)}$

basis:

	$p\ \mathbf{en}\ q$	, premise
	$p \wedge \neg q\ \mathbf{co}\ p \vee q$	, from above
	$r\ \mathbf{co}\ s$	, premise
	$p \wedge \neg q \wedge r\ \mathbf{co}\ (p \wedge s) \vee (q \wedge s)$	, conjunction of the above two
	$p \wedge \neg q \wedge r\ \mathbf{co}\ (p \wedge s) \vee (q \wedge (r \vee (\neg r \wedge s)))$	, in the rhs, write s as $r \vee (\neg r \wedge s)$ using $r \Rightarrow s$ (because $r\ \mathbf{co}\ s$)
(1)	$p \wedge \neg q \wedge r\ \mathbf{co}\ (p \wedge s) \vee (q \wedge r) \vee (\neg r \wedge s)$	, weakening the rhs
	$p \wedge \neg q$ transient	, from the premise $p\ \mathbf{en}\ q$
	$p \wedge \neg q \wedge r$ transient	, strengthening the above (see Section 4.3.4)
	$p \wedge s\ \mathbf{en}\ (q \wedge r) \vee (\neg r \wedge s)$	, from (1) and the above using the definition of **en** (use $r \Rightarrow s$)
	$p \wedge s \mapsto (q \wedge r) \vee (\neg r \wedge s)$	, basis rule for $\mapsto$

transitivity:

There is a predicate b such that $p \mapsto b \mapsto q$. Hence,

	$r\ \mathbf{co}\ s$	, premise
(2)	$p \wedge s \mapsto (b \wedge r) \vee (\neg r \wedge s)$	, induction hypothesis on $p \mapsto b$ and $r\ \mathbf{co}\ s$
	$b \wedge s \mapsto (q \wedge r) \vee (\neg r \wedge s)$	, induction hypothesis on $b \mapsto q$ and $r\ \mathbf{co}\ s$
	$b \wedge r \mapsto (q \wedge r) \vee (\neg r \wedge s)$	, lhs strengthening using $r \Rightarrow s$
	$p \wedge s \mapsto (q \wedge r) \vee (\neg r \wedge s)$	, cancellation on (2) and above

disjunction:

There is a set, S, of predicates such that $b \mapsto q$ for every b in S and $p = \langle \exists\ b\ :\ b \in S\ :\ b \rangle$. Hence, for b in S

$b \mapsto q$, premise
r **co** s , premise
$b \wedge s \mapsto (q \wedge r) \vee (\neg r \wedge s)$, induction hypothesis
$\langle \exists\ b : b \in S : b \wedge s \rangle \mapsto (q \wedge r) \vee (\neg r \wedge s)$
, disjunction
$\langle \exists\ b : b \in S : b \rangle \wedge s \mapsto (q \wedge r) \vee (\neg r \wedge s)$
, predicate calculus
$p \wedge s \mapsto (q \wedge r) \vee (\neg r \wedge s)$, $p = \langle \exists\ b : b \in S : b \rangle$ □

- (induction) $$\frac{\langle \forall\ m :: p \wedge M = m \mapsto (p \wedge M < m) \vee q \rangle}{p \mapsto q}$$

where M is a total function from program states to W, and $(W, <)$ is well-founded. The premise is over all m in W.

The premise can be written, for all m in W,

$$p \wedge M = m \mapsto \langle \exists\ n : n < m : p \wedge M = n \rangle \vee q$$

In the rest of this proof m, n are quantified over W. We use the abbreviation

$$A.n \triangleq p \wedge M = n \mapsto q$$

Now, for any m in W

$\langle \forall\ n : n < m : A.n \rangle$
$\Rightarrow$ {expanding $A.n$ using its definition}
$\langle \forall\ n : n < m : p \wedge M = n \mapsto q \rangle$
$\Rightarrow$ {disjunction}
$\langle \exists\ n : n < m : p \wedge M = n \rangle \mapsto q$
$\Rightarrow$ {cancellation using the premise: $p \wedge M = m \mapsto \langle \exists\ n : n < m : p \wedge M = n \rangle \vee q$}
$p \wedge M = m \mapsto q$
$\Rightarrow$ {using the abbreviation $A.m$ for the above}
$A.m$

We have thus established for any m in W that
$\langle \forall\ n : n < m : A.n \rangle \Rightarrow A.m$
Applying the induction principle (on well-founded sets) we can conclude,
$\langle \forall\ m :: A.m \rangle$.

Now,
$\langle \forall\ m :: A.m \rangle$
$\Rightarrow$ {expanding $A.m$}
$\langle \forall\ m :: p \wedge M = m \mapsto q \rangle$
$\Rightarrow$ {disjunction and that p does not name m}

$p \ \wedge \ \langle \exists \ m \ :: \ M = m \rangle \ \mapsto \ q$
$\Rightarrow$ {range of total function M is W, so $\langle \exists \ m \ :: \ M = m \rangle = true$}
$p \ \mapsto \ q$ $\Box$

- (completion)

$$\frac{\begin{array}{l}\langle \forall \ i \ : \ 0 \leq i < N \ : \\ \quad p_i \ \mapsto \ q_i \ \vee \ b \\ \quad q_i \ \textbf{co} \ q_i \ \vee \ b \\ \rangle\end{array}}{\langle \forall \ i \ : \ 0 \leq i < N \ : \ p_i \rangle \ \mapsto \ \langle \forall \ i \ : \ 0 \leq i < N \ : \ q_i \rangle \ \vee \ b}$$

We will prove the result only for the special case, $b = false$. The proof of the general case is left to Exercise 17. Thus, we have to show,

$$\frac{\begin{array}{l}\langle \forall \ i \ : \ 0 \leq i < N \ : \\ \quad (1) \quad p_i \ \mapsto \ q_i \\ \quad (2) \quad q_i \text{ stable} \\ \rangle\end{array}}{\langle \forall \ i \ : \ 0 \leq i < N \ : \ p_i \rangle \ \mapsto \ \langle \forall \ i \ : \ 0 \leq i < N \ : \ q_i \rangle}$$

Proof is by induction on N.

$N = 0$: $true \ \mapsto \ true$, follows from the implication rule for $\mapsto$
$N + 1$: The goal is to show
$\langle \forall \ i \ : \ 0 \leq i < N + 1 \ : \ p_i \rangle \ \mapsto \ \langle \forall \ i \ : \ 0 \leq i < N + 1 \ : \ q_i \rangle$

Writing P for $\langle \forall \ i \ : \ 0 \leq i < N \ : \ p_i \rangle$ and Q for $\langle \forall \ i \ : \ 0 \leq i < N \ : \ q_i \rangle$, we have to show,

$$P \ \wedge \ p_N \ \mapsto \ Q \ \wedge \ q_N$$

Essential to the proof is the notion of wlt (Section 4.6.1). From $p_N \ \mapsto \ q_N$, there is a predicate r (r is $wlt.q_N$) such that,

(3) $r \ \mapsto \ q_N$, from W3 of Sec. 4.6.1
(4) $p_N \ \Rightarrow \ r$, from $p_N \ \mapsto \ q_N$ and W2 of Sec. 4.6.1
(5) $q_N \ \Rightarrow \ r$, from W4 of Sec. 4.6.1
(6) $r \ \wedge \ \neg q_N \ \textbf{co} \ r$, from W5 of Sec. 4.6.1

We have,
q_N stable , from (2)
$r \ \vee \ q_N$ stable , disjunction of above and (6)
(7) r stable , above and (5)

The main proof is as follows.

$$
\begin{array}{ll}
& P \ \wedge \ p_N \\
\Rightarrow & \{p_N \ \Rightarrow \ r \qquad\qquad \text{, from (4)}\} \\
& P \ \wedge \ r \\
\mapsto & \{P \ \mapsto \ Q \qquad\qquad \text{, induction hypothesis} \\
& \ \ r \text{ stable} \qquad\qquad \text{, from (7)} \\
& \ \ P \ \wedge \ r \ \mapsto \ Q \ \wedge \ r \quad \text{, stable conjunction, see Sec. 4.4.5}\} \\
& Q \ \wedge \ r \\
\mapsto & \{r \ \mapsto \ q_N \qquad\qquad \text{, from (3)} \\
& \ \ Q \text{ stable} \qquad\qquad \text{, conjunction over (2)} \\
& \ \ Q \ \wedge \ r \ \mapsto \ Q \ \wedge \ q_N \ \text{, stable conjunction, see Sec. 4.4.5}\} \\
& Q \ \wedge \ q_N
\end{array}
$$

□

4.4.5 Corollaries of the Derived Rules

The following corollary, like its namesake for **co**, permits us to conjoin a stable predicate to both sides of a *leads-to* property.

- (stable conjunction)
$$\frac{p \ \mapsto \ q \ , \ r \quad \text{stable}}{p \ \wedge \ r \ \mapsto \ q \ \wedge \ r}$$

Proof: Set s to r in the PSP rule. □

- (Corollary of PSP) Given r **co** s we have $r \ \Rightarrow \ s$. Therefore, strengthening the lhs (by replacing s by r) and/or weakening the rhs (by replacing r by s) in the consequent of the PSP rule gives the following results:

$$\frac{\begin{array}{c} p \ \mapsto \ q \\ r \ \textbf{co} \ s \end{array}}{\begin{array}{ll} p \ \wedge \ r \ \mapsto \ (q \ \wedge \ r) \ \vee \ (\neg r \ \wedge \ s) & \text{, lhs strengthening} \\ p \ \wedge \ s \ \mapsto \ (q \ \wedge \ s) \ \vee \ (\neg r \ \wedge \ s) & \text{, rhs weakening} \\ p \ \wedge \ r \ \mapsto \ (q \ \wedge \ s) \ \vee \ (\neg r \ \wedge \ s) & \text{, both of the above} \end{array}}$$

- (Corollary of Induction) In the following corollary the range of m (in the premise of the induction rule) is restricted to a predicate $r.m$. Suppose "$<$" is a well-founded order over r.

$$\frac{\begin{array}{l} \langle \forall \ m \ : \ r.m \ : \\ \qquad p \ \wedge \ M = m \ \mapsto \ (p \ \wedge \ M < m) \ \vee \ q \\ \rangle \end{array}}{p \ \mapsto \ (p \ \wedge \ \neg r.M) \ \vee \ q}$$

Proof:

$$p \ \wedge \ M = m \ \wedge \ r.m \ \mapsto \ (p \ \wedge \ M < m) \ \vee \ q \qquad \text{, premise}$$

$$
\begin{array}{ll}
p \ \wedge \ M = m \ \wedge \ r.M \ \mapsto \ (p \ \wedge \ M < m) \ \vee \ q & \text{, rewriting lhs} \\
p \ \wedge \ M = m \ \wedge \ \neg r.M \ \mapsto \ p \ \wedge \ \neg r.M & \text{, implication} \\
p \ \wedge \ M = m \ \mapsto \ (p \ \wedge \ M < m) \ \vee \ (p \ \wedge \ \neg r.M) \ \vee \ q & \\
 & \text{, disjunction} \\
p \ \mapsto \ (p \ \wedge \ \neg r.M) \ \vee \ q & \text{, induction} \qquad \Box
\end{array}
$$

An important special case arises when M is integer valued, $r.m$ is of the form $m > L$, for some lower bound L, and "$<$" is the standard less-than relation over integers. Then, we conclude from the above corollary

- (induction over integers)

$$
\frac{\begin{array}{l}\langle \forall \ m \ : \ m > L \ : \\ \qquad p \ \wedge \ M = m \ \mapsto \ (p \ \wedge \ M < m) \ \vee \ q \\ \rangle\end{array}}{p \ \mapsto \ (p \ \wedge \ M \leq L) \ \vee \ q}
$$

If $r.m$ imposes no lower bound on m—for instance, $r.m \ \equiv \ \mathit{true}$—then the conclusion holds for any L.

Using "greater-than" in place of "less-than," we have the analogous

- (induction over integers)

$$
\frac{\begin{array}{l}\langle \forall \ m \ : \ m < L \ : \\ \qquad p \ \wedge \ M = m \ \mapsto \ (p \ \wedge \ M > m) \ \vee \ q \\ \rangle\end{array}}{p \ \mapsto \ (p \ \wedge \ M \geq L) \ \vee \ q}
$$

A simple special case of the induction rule is

- (simple corollary of induction)

$$
\frac{\begin{array}{l}\langle \forall \ m \ :: \\ \qquad p \ \wedge \ M = m \ \mapsto \ M < m \\ \rangle\end{array}}{\mathit{true} \ \mapsto \ \neg p}
$$

4.5 Applications

We consider a few small examples in which we employ the progress operators for specifications and deductions of program properties. The emphasis is on converting verbal descriptions to formal specifications and using the derived rules for deductions. In most cases, we also supply the typical verbal arguments that justify the deductions and contrast them with formal proofs.

4.5.1 Nonoperational Descriptions of Algorithms

We consider the algorithm for computing the maximum of a nonempty set of numbers, S, as described in earlier. The safety properties that we had postulated are as follows. Here, v is the variable in which the maximum is being computed and m is any integer.

$$\textit{initially } v = -\infty \tag{ND1}$$
$$v = m \textbf{ co } v = m \ \vee \ (v \in S \ \wedge \ v > m) \tag{ND2}$$

We derived a number of safety properties including

$$v \leq M \tag{ND3}$$

where M is the maximum in S, i.e., $M = \langle \max x \ : \ x \in S \ : \ x \rangle$. Now, we add the following progress property, where x is free.

$$m \in S \ \mapsto \ v \geq m \tag{ND4}$$

which says that eventually v is at least m for any m in S. It is easy to establish that v will equal M eventually.

Proof of $\textit{true} \ \mapsto \ v = M$:

$m \in S \ \mapsto \ v \geq m$	, (ND4)
$M \in S \ \mapsto \ v \geq M$	, instantiating m by M
$\textit{true} \ \mapsto \ v \geq M$	, substitution axiom on the lhs (S is nonempty)
$\textit{true} \ \mapsto \ v = M$	, substitution axiom, conjoining (ND3) on the rhs

A refinement of this algorithm was considered earlier. Let S be represented by an array, A, and s be an index into A. We had

$$\textit{initially } v, s = -\infty, 0 \text{ and} \tag{ND5}$$
$$v = \langle \max i \ : \ 0 \leq i < s \ : \ A[i] \rangle \tag{ND6}$$

We will show that v acquires the maximum value in A given that the index s increases eventually as long as the end of the array is not reached. Formally, for all k, $0 \leq k < N$ (where A has N, $N \geq 0$, elements), assume

$$s = k \ \mapsto \ s = k + 1 \tag{ND7}$$

Proof of initial condition $\mapsto \ v = \langle \max i \ : \ 0 \leq i < N \ : \ A[i] \rangle$:

$s \leq N \ \mapsto \ s = N$	, from (ND7) (see Exercise 14a)
$s = 0 \ \mapsto \ s = N$	, strengthen lhs of the above using $N \geq 0$
$s = 0 \ \mapsto \ v = \langle \max i \ : \ 0 \leq i < N \ : \ A[i] \rangle$	, conjoin (ND6) to rhs using substitution axiom.
initial condition $\mapsto \ v = \langle \max i \ : \ 0 \leq i < N \ : \ A[i] \rangle$	, strengthen lhs using (ND5)

4.5.2 Common Meeting Time

The Common Meeting Time problem was discussed earlier. We had functions f, g that mapped nonnegative reals to nonnegative reals. Now, we tighten the requirements on f, g: These functions map natural numbers to natural numbers. Thus, for all natural m, n

$$\begin{array}{l} m \leq n \;\Rightarrow\; f(m) \leq f(n) \\ m \leq n \;\Rightarrow\; g(m) \leq g(n) \end{array} \qquad \text{(CMT1)}$$

A variable t—previously of type real, now of type natural number—is postulated to satisfy

$$\textit{initially}\ t = 0 \qquad \text{(CMT2)}$$

$$t = m \ \mathbf{co}\ t \leq \max(f(m), g(m)) \qquad \text{(CMT3)}$$

We had established earlier, from CMT1– CMT3, that t exceeds no common meeting time:

$$com(n) \;\Rightarrow\; t \leq n \qquad \text{(CMT4)}$$

The essential safety property, (CMT3), can be implemented by program *skip*, that does not change t. In order to guarantee that t eventually equals the earliest common meeting time, we add a progress requirement: If t is not a common meeting time then it increases eventually.

$$\neg com(t) \;\wedge\; t = m \;\mapsto\; t > m \qquad \text{(CMT6)}$$

We show that eventually t will be equal to a common meeting time, if there is one.

$$\langle \exists\ n \;::\; com(n) \rangle \;\mapsto\; com(t) \qquad \text{(CMT7)}$$

Together (CMT4) and (CMT7) imply that if there is a common meeting time, t will eventually equal the earliest common meeting time, if one exists.

Proof of (CMT7): For natural numbers m, n

$\neg com(t) \;\wedge\; t = m \;\mapsto\; t > m$	, rewrite (CMT6)
$com(t) \;\wedge\; t = m \;\mapsto\; com(t)$	, implication
$t = m \;\mapsto\; t > m \;\vee\; com(t)$	, disjunction of the above two
$true \;\mapsto\; t > n \;\vee\; com(t)$	, induction on integers
$com(n) \;\mapsto\; (com(n) \;\wedge\; t > n) \;\vee\; com(t)$	, stable conjunction with $com(n)$
$com(n) \;\mapsto\; com(t)$	, use CMT4 to cancel the first disjunct in the rhs
$\langle \exists\ n \;::\; com(n) \rangle \;\mapsto\; com(t)$	, disjunction over all n. □

This proof is invalid if t is of type real because the induction step in the above proof is then invalid. The reader can construct a counterexample to (CMT7) by having t increase extremely slowly, say, by $1/2^i$ in the i^{th} step.

One way to implement the progress condition, CMT6, is to increment t by 1 in each step and check to see if $com(t)$ holds. The monotonicity condition on f, g – given by CMT1– is far too weak to permit many other strategies. If the functions are also ascending, i.e., for all natural n

$$n \leq f(n),\ n \leq g(n)$$

then the programs (P1,P2) given earlier satisfy

$$\neg com(t) \wedge t = m \ \ \mathbf{en} \ \ t > m$$

and, hence, CMT6 as well.

4.5.3 Token Ring

We have considered the properties of a token ring earlier. The following safety properties were postulated there.

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.

$$\begin{array}{ll} initially\ e_i \ \Rightarrow \ p = i & \text{(TR0)} \\ e_i \ \mathbf{co} \ e_i \ \vee \ t_i & \text{(TR1)} \\ t_i \ \mathbf{co} \ t_i \ \vee \ h_i & \text{(TR2)} \\ h_i \ \mathbf{co} \ h_i \ \vee \ e_i & \text{(TR3)} \\ h_i \ \wedge \ p \neq i \ \mathbf{co} \ h_i & \text{(TR4)} \\ p = i \ \mathbf{co} \ p = i \ \vee \ \neg e_i & \text{(TR5)} \end{array}$$

It was possible earlier to deduce mutual exclusion, a safety property, from (TR0–TR5). Now, we postulate some progress properties and establish the absence of starvation.

First, we require that a hungry token-holder transit to eating.

$$h_i \ \wedge \ p = i \ \mapsto \ e_i \qquad \text{(TR6)}$$

Next, we require that the token move from the current token-holder to its right neighbor. In the following, i' is the right neighbor of i.

$$p = i \ \mapsto \ p = i' \qquad \text{(TR7)}$$

Note that (TR7) does not require the token to go from i to i' directly.

We establish absence of starvation for all processes. That is, we show for any j, $0 \leq j < N$,

$$h_j \;\mapsto\; e_j \qquad \text{(TR8)}$$

Proof of (TR8): Consider an arbitrary j, $0 \leq j < N$. First, we show that j eventually holds the token, i.e.,

$$true \;\mapsto\; p = j \qquad \text{(TR9)}$$

The proof is by induction over (TR7). To apply the induction rule we define, $i' \prec i$ for all process indices i, where $i' \neq j$. The relation $\prec$ induces a linear order over the processes,

$$\ldots i' \prec i \ldots \prec j' \prec j.$$

We can then rewrite (TR7) as
$p = i \;\mapsto\; p \prec i$, for all i where $i' \neq j$.
$p = i \;\mapsto\; p = j$, for i where $i' = j$.
Combining the above two

$$\langle \forall\; i \;::\; p = i \;\mapsto\; p \prec i \;\vee\; p = j \rangle$$

Apply the induction rule (since $\prec$ is a linear order over a finite set) to get (TR9). Now, we are ready to prove (TR8).

h_j **co** $h_j \;\vee\; e_j$	, from (TR3) using j for i
$true \;\mapsto\; p = j$	, (TR9)
$h_j \;\mapsto\; (h_j \;\wedge\; p = j) \;\vee\; e_j$	, PSP {use $(\neg h_j \;\wedge\; e_j) = e_j$}
$h_j \;\wedge\; p = j \;\mapsto\; e_j$	, from (TR6) using j for i
$h_j \;\mapsto\; e_j$	, cancellation on the above two

The standard verbal argument for this problem follows the above proof steps closely. The formalism allows us to combine a few special cases. The role of induction—if every process relinquishes the token eventually, then every process acquires the token eventually—is made explicit in our proof. Note that the proof is entirely independent of the kind of fairness assumed in (TR6) and (TR7).

4.5.4 Unordered Channel

We consider a channel, directed from one process to another, along which messages are sent from the former to the latter. It is required to design a protocol by which every message sent is delivered eventually, though the order of delivery may be different from the order of transmission. The following scheme implements the protocol. Every message sent along the channel is assigned a *sequence number* (a natural number) and the sequence numbers are strictly increasing in the order of transmission. A message that has the lowest sequence number in the channel at any point in the computation is eventually delivered. We claim that this scheme guarantees

delivery of every message sent. Note, however, that the delivery order may not be monotonic in sequence numbers.

The proposed scheme can be described by the following properties. In this description, s is the set of sequence numbers of the messages in the channel, and x, y are arbitrary sequence numbers. The lowest sequence number in s is $\min.s$, where for an empty set s we take $\min.s$ to be ∞. If $\min.s$ has a value x then $\min.s$ cannot decrease as a result of adding a number to s, from the monotonicity of sequence numbers. Also, removing a number from s does not decrease $\min.s$. Hence, the safety property, UC1, says that $\min.s$ never decreases as long as s is nonempty. The progress property, UC2, says that the smallest element of s is eventually removed from s. Our goal, UC3, is to show that every sequence number is eventually outside s. We assume throughout that s is a finite set.

$$\min.s = x \ \textbf{co} \ \min.s \geq x \qquad \text{(UC1)}$$
$$\min.s = x \ \mapsto \ x \notin s \qquad \text{(UC2)}$$

Show
$$\mathit{true} \ \mapsto \ y \notin s \qquad \text{(UC3)}$$

Our proof formalizes the following argument. From UC1, $\min.s$ never decreases. From UC2, if $x = \min.s$ then x is eventually removed; therefore, $\min.s$ increases eventually. It follows that $\min.s$ increases without bound as long as s is nonempty; hence, every number is eventually outside s.

$\min.s = x \ \mapsto \ (x \notin s \ \wedge \ \min.s \geq x) \ \vee \ (\min.s \neq x \ \wedge \ \min.s \geq x)$
, PSP on UC1, UC2
$\min.s = x \ \mapsto \ \min.s > x$, simplifying the rhs
$\mathit{true} \ \mapsto \ \min.s > y$, induction on integers (using y as the upper bound)
$\mathit{true} \ \mapsto \ y \notin s$, weakening the rhs using $\min.s > y \ \Rightarrow \ y \notin s$

4.5.5 Dynamic Graphs

In the problem treated earlier, a finite directed graph could be modified by directing all incident edges on a node towards that node. The effect of this operation was to make that node a bottom node. We showed that this operation does not create new paths to non-bottom nodes, i.e., for arbitrary nodes u, v, with $u \ R \ v$ denoting that there is a path from u to v and $v.\ \bot$ denoting that v is bottom

$$\neg u \ R \ v \ \textbf{co} \ \neg u \ R \ v \ \vee \ v.\ \bot \qquad \text{(G1)}$$

We showed that one of the consequences of (G1) is that once the graph becomes acyclic, it remains acyclic. We assume, henceforth, that the graph is initially acyclic; therefore, it is always acyclic, i.e., for all u

$$\neg u \ R \ u \qquad \text{(G2)}$$

Now, we add a progress requirement. Call a node a *top* node if it has no incoming edge (recall that a *bottom* node has no outgoing edge; an isolated node is both top and bottom). Observe that a top node remains a top node as long as edge redirection is not applied to it. We require that edge redirection be applied to every top node eventually; the effect is to make the node a bottom node. Therefore, using $v.\top$ to denote that v is a top node, we require

$$v.\top \;\mapsto\; v.\bot \tag{G3}$$

We will show that every node eventually becomes a bottom node, i.e.,

$$\mathit{true} \;\mapsto\; u.\bot \tag{G4}$$

The main arguments behind the proof of (G4) are as follows. For any node u, consider the set of *ancestors* of u, i.e., the nodes that have paths to u. This set does not grow as long as u is non-bottom, a safety property that we will establish from G1. Next, since the graph is acyclic, either u is top or u has an ancestor, v, that is top. In the first case, from G3, u will become bottom. In the second case, from G3, v will become bottom and therefore cease to be an ancestor of u; hence, the ancestor set of u decreases in size eventually. Since the ancestor set cannot decrease forever, u will become top and then, from G3, eventually become bottom.

In the following, $u.an$ denotes the ancestor set of u, i.e.,

$$v \in u.an \;\equiv\; v \;R\; u \tag{G5}$$

We will use the following facts about top and bottom. A node is top iff its ancestor set is empty (G6) and a bottom node does not belong to any ancestor set (G7).

$$u.\top \;\equiv\; u.an = \phi \tag{G6}$$
$$v.\bot \;\Rightarrow\; v \notin u.an \tag{G7}$$

First, we establish that the ancestor set does not grow as long as a node remains non-bottom. In the rest of this proof, S is any fixed set of nodes.

Lemma: $\quad u.an = S \;\mathbf{co}\; u.an \subseteq S \;\vee\; u.\bot$ (G8)

Proof:

$\neg v \;R\; u \;\mathbf{co}\; \neg v \;R\; u \;\vee\; u.\bot$	, G1 with u, v interchanged
$v \notin u.an \;\mathbf{co}\; v \notin u.an \;\vee\; u.\bot$	, using G5 to rewrite $\neg v \;R\; u$
$\langle \forall\, v \;:\; v \notin S \;:\; v \notin u.an \rangle \;\mathbf{co}\; \langle \forall\, v \;:\; v \notin S \;:\; v \notin u.an \rangle \;\vee\; u.\bot$	, conjunction over all v, $v \notin S$
$u.an \subseteq S \;\mathbf{co}\; u.an \subseteq S \;\vee\; u.\bot$	, simplifying the two sides
$u.an = S \;\mathbf{co}\; u.an \subseteq S \;\vee\; u.\bot$	, strengthening left side $\Box$

Lemma: (G4) $\mathit{true} \mapsto u.\ \bot$

Proof:

$v.\top \mapsto v.\ \bot$, G3

$u.an = S \ \wedge\ v.\top \mapsto v \notin u.an$, strengthening lhs and weakening rhs (using G7)

$u.an = S \ \wedge\ v.\top \ \wedge\ v \in S \mapsto v \notin u.an \ \wedge\ v \in S$
, stable conjunction with $v \in S$, where S is a subset of nodes

$u.an = S \ \wedge\ v.\top \ \wedge\ v \in S \mapsto u.an \neq S$
, weakening rhs

$u.an = S \ \wedge\ \langle \exists\ v \ ::\ v.\top \ \wedge\ v \in S \rangle \mapsto u.an \neq S$
, disjunction over v

$u.an = S \ \wedge\ S \neq \phi \mapsto u.an \neq S$, an acyclic graph S has a top node $\equiv\ (S \neq \phi)$

$u.an = S$ **co** $u.an \subseteq S \ \vee\ u.\ \bot$, from Lemma (G8)

$u.an = S \ \wedge\ S \neq \phi \mapsto u.an \subset S \ \vee\ u.\ \bot$
, PSP and weakening rhs

$u.an = S \ \wedge\ S = \phi \mapsto u.\ \bot$, $u.an = \phi \ \Rightarrow\ \{\text{G6}\}\ u.\top \mapsto \{\text{G3}\}\ u.\ \bot$

$u.an = S \mapsto u.an \subset S \ \vee\ u.\ \bot$, disjunction of the above two

$\mathit{true} \mapsto u.\ \bot$, induction (subset relation over finite sets is well founded) □

4.5.6 Strong Semaphore: An Example of Strong Fairness

The standard example to illustrate the concept of strong fairness is a binary semaphore that is shared by two processes. Let x, y denote the number of times that the two processes, respectively, are granted the semaphore (i.e., successfully complete their P-operations on the semaphore). We assume that each process releases the semaphore (i.e., performs a V-operation) eventually. An outline of the program is as follows.

$$
\begin{array}{llll}
 & \alpha & :: & s, x := \mathit{false}, x+1 \quad \text{if} \quad s \\
[\!] & \beta & :: & s, y := \mathit{false}, y+1 \quad \text{if} \quad s \\
[\!] & \gamma & :: & s := \mathit{true}
\end{array}
$$

Under the weak fairness assumption we can establish only that $x + y$ increases without bound; we can make no such guarantee for either x or y. Now, we impose the following strong fairness condition for statement α: If the guard of α (i.e., s) is infinitely often *true* then α is executed infinitely often. We assume weak fairness for the remaining statements, β, γ. Our goal is to show that x increases without bound under this strong fairness condition.

The strong fairness condition can be added as an axiom to the program; for any integer k

$$\text{(SF)} \qquad (\mathit{true} \mapsto s) \ \Rightarrow\ (x = k \mapsto x = k+1)$$

So, we regard our system as consisting of the program (with weak fairness condition) plus the axiom SF. It is then straightforward to show that for

any integer m

$$true \;\mapsto\; x > m$$

The proof is as follows.

$true$ **en** s	, from the program text
$true \;\mapsto\; s$	, definition of $\mapsto$
$x = k \;\mapsto\; x = k + 1$	, from above using SF
$true \;\mapsto\; x > m$	, induction on integers

This treatment of strong fairness, though sound, is incomplete. To see this we consider the following program where x, y are integers and b is boolean, and all statements are executed under strong fairness.

$$\begin{array}{lllll} & \alpha & :: & x := x+1 & \text{if} \quad b \\ [\!] & \beta & :: & y := y+1 & \text{if} \quad \neg b \\ [\!] & \gamma & :: & b := \neg b & \text{if} \quad x = y \end{array}$$

We see that $x + y$ is guaranteed to increase by the following operational argument. Both x, y are nondecreasing. In any execution either b holds infinitely often or $\neg b$ holds infinitely often; therefore, either x or y increases infinitely often and hence, $x + y$ increases without bound. Unfortunately, our proof method cannot be used to prove this result; adding the following strong fairness conditions to the program

$(true \;\mapsto\; b) \;\Rightarrow\; (x = k \;\mapsto\; x = k + 1)$ and
$(true \;\mapsto\; \neg b) \;\Rightarrow\; (y = n \;\mapsto\; y = n + 1)$

does not help us to prove that $x+y$ increases. This is because we can prove neither

$$true \;\mapsto\; b \quad \text{nor} \quad true \;\mapsto\; \neg b$$

(because, if initially $x, y, b = 0, 1, false$ then $(x < y) \;\wedge\; \neg b$ persists forever; similarly with $x, y, b = 1, 0, true$ as the initial condition $(x > y) \;\wedge\; b$ persists). The trouble is

$$(true \;\mapsto\; b) \;\Rightarrow\; (x = k \;\mapsto\; x = k + 1)$$

merely says that if b holds infinitely often in *all* execution sequences then x will be incremented infinitely often in *all* execution sequences. What we need is a stronger statement: In any execution sequence, if b holds infinitely often then x is incremented infinitely often (in that sequence). Our logic does not permit us to say this.

4.6 Theoretical Issues

We touch upon a few theoretical issues concerning *leads-to*. In Section 4.6.1 we show the existence of the "weakest predicate that *leads-to*" q, for any q, and we prove some of its properties. (These properties have been used in proving the completion rule in Section 4.4.4) We give a fixpoint characterization of this predicate in Section 4.6.2 The role of the disjunction rule is examined in Section 4.6.3; We show that the validity of disjunction over a finite set of predicates is derivable from the basis and transitivity rules; therefore, the main use of the disjunction rule is in its application to an infinite set of predicates.

4.6.1 wlt

For a predicate q, let *wlt.q* be the weakest predicate that *leads-to* q. The definition of *wlt* is

$$wlt.q \;\triangleq\; \langle \exists\; p \;:\; p \;\mapsto\; q \;:\; p \rangle \tag{W1}$$

We show that *wlt.q* is indeed the weakest predicate leading to q, i.e.,

$$p \;\mapsto\; q \;\equiv\; (p \;\Rightarrow\; wlt.q) \tag{W2}$$

To see this, we first prove $(p \;\mapsto\; q) \;\Rightarrow\; (p \;\Rightarrow\; wlt.q)$

$p \;\mapsto\; q$	, assume
$p \;\Rightarrow\; wlt.q$	, from (W1)

Conversely, we show $(p \;\Rightarrow\; wlt.q) \;\Rightarrow\; (p \;\mapsto\; q)$

$\langle \forall\; r \;:\; r \;\mapsto\; q \;:\; r \;\mapsto\; q \rangle$	, trivially
$\langle \exists\; r \;:\; r \;\mapsto\; q \;:\; r \rangle \;\mapsto\; q$	, disjunction
$wlt.q \;\mapsto\; q$	, using (W1)
$p \;\mapsto\; q$	, strengthening the lhs using $p \;\Rightarrow\; wlt.q$

This concludes the proof of (W2). We can use (W2) to answer if $p \;\mapsto\; q$ for arbitrary p, q, by answering if $p \;\Rightarrow\; wlt.q$. This is the preferred method when *wlt.q* can be computed efficiently.

As special cases of (W2), substituting *wlt.q* for p and using $wlt.q \;\Rightarrow\; wlt.q$

$$wlt.q \;\mapsto\; q \tag{W3}$$

Also, substituting q for p in (W2) and using $q \;\mapsto\; q$,

$$q \;\Rightarrow\; wlt.q \tag{W4}$$

Next, we show

$$wlt.q \ \wedge \ \neg q \ \mathbf{co} \ wlt.q \qquad \text{(W5)}$$

Proof of (W5): $wlt.q \ \wedge \ \neg q \ \mathbf{co} \ wlt.q$

Given $p \ \mapsto \ q$, for any p, q, we show that there is a predicate b satisfying

1. $p \ \Rightarrow \ b$
2. $b \ \mapsto \ q$
3. $b \ \wedge \ \neg q \ \mathbf{co} \ b \ \vee \ q$

We establish (W5) from (1,2,3), as follows.

$wlt.q \ \mapsto \ q$, from (W3)

Using (1,2,3), with p as $wlt.q$ and q as q, there is a predicate w such that

$1'$. $wlt.q \ \Rightarrow \ w$
$2'$. $w \ \mapsto \ q$
$3'$. $w \ \wedge \ \neg q \ \mathbf{co} \ w \ \vee \ q$

Now,

$w \ \Rightarrow \ wlt.q$	, from ($2'$ and W2)
$w \ \equiv \ wlt.q$	, from above and $1'$
$wlt.q \ \wedge \ \neg q \ \mathbf{co} \ wlt.q \ \vee \ q$	, from $3'$ replacing w by $wlt.q$
$wlt.q \ \wedge \ \neg q \ \mathbf{co} \ wlt.q$	, simplifying rhs using (W4)

This establishes (W5). Now, we prove (1,2,3). The proof is by induction on the structure of the proof of $p \ \mapsto \ q$.

(basis) $p \ \mathbf{en} \ q$

Let $b \triangleq p$. Then,

$p \ \Rightarrow \ b$	, trivially, from $b = p$
$b \ \mapsto \ q$	, from $p \ \mapsto \ q$ and $b = p$
$b \ \wedge \ \neg q \ \mathbf{co} \ b \ \vee \ q$	, from $p \ \wedge \ \neg q \ \mathbf{co} \ p \ \vee \ q$ and $b = p$

(transitivity) $p \ \mapsto \ r \ \mapsto \ q$, for some predicate r

By induction hypothesis on $p \ \mapsto \ r$, there is a predicate br

$p \ \Rightarrow \ br$, $br \ \mapsto \ r$, $br \ \wedge \ \neg r \ \mathbf{co} \ br \ \vee \ r$

Similarly, there is a predicate bq for $r \ \mapsto \ q$ where

$r \ \Rightarrow \ bq$, $bq \ \mapsto \ q$, $bq \ \wedge \ \neg q \ \mathbf{co} \ bq \ \vee \ q$

Define $b \ \triangleq \ br \ \vee \ bq$

- $p \ \Rightarrow \ b$, from $p \ \Rightarrow \ br \ \Rightarrow \ br \ \vee \ bq$
- $b \ \mapsto \ q$

$br \ \mapsto \ r$	, given
$bq \ \mapsto \ q$	, given
$br \ \vee \ bq \ \mapsto \ r \ \vee \ q$	, disjunction
$b \ \mapsto \ r \ \vee \ q$	, definition of b

$b \mapsto q$, cancellation using $r \mapsto q$

- $b \wedge \neg q$ **co** $b \vee q$

  $br \wedge \neg r$ **co** $br \vee r$, given

  $br \wedge \neg bq$ **co** $br \vee r$, strengthening lhs using $r \Rightarrow bq$

  $br \wedge \neg bq \wedge \neg q$ **co** $br \vee r$, strengthening lhs

  $bq \wedge \neg q$ **co** $bq \vee q$, given

  $(br \vee bq) \wedge \neg q$ **co** $br \vee bq \vee q$, disjunction; in the rhs $r \Rightarrow bq$

  $b \wedge \neg q$ **co** $b \vee q$, $b \equiv br \vee bq$

(disjunction) $r \mapsto q$, for r in some set S, and $p = \langle \exists\ r\ :\ r \in S\ :\ r \rangle$

By induction hypothesis, for every r in S, there is a predicate $b.r$,
$r \Rightarrow b.r$, $b.r \mapsto q$, $b.r \wedge \neg q$ **co** $b.r \vee q$
Define $b \triangleq \langle \exists\ r\ :\ r \in S\ :\ b.r \rangle$. For any r, $r \in S$,

- $p \Rightarrow b$

  $r \Rightarrow b.r$, given

  $p \Rightarrow b$, disjunction over all r in S

- $b \mapsto q$

  $b.r \mapsto q$, given

  $\langle \exists\ r\ :\ r \in S\ :\ b.r \rangle \mapsto q$, disjunction

  $b \mapsto q$, definition of b

- $b \wedge \neg q$ **co** $b \vee q$

  $br \wedge \neg q$ **co** $b.r \vee q$, given

  $b \wedge \neg q$ **co** $b \vee q$, taking disjunction over all r in S □

4.6.2 A Fixpoint Characterization of wlt

The following fixpoint characterization of *wlt* is under weak fairness. We define a predicate transformer *we*, that captures the essence of *ensures* and we define *wlt* in terms of *we*. These definitions have been found useful in automatic verifications of finite state programs where the extreme solutions (i.e., weakest or strongest) can be computed iteratively.

Predicate $we.p$ is the weakest predicate such that, starting in any state satisfying $we.p$, eventually p is established, and $we.p$ holds until p is established. The predicate $we.p$ can be written as a disjunction of several predicates, $(we.p)_t$, one predicate for each transition t in the given program. Predicate $(we.p)_t$ has the same meaning as $we.p$, but the additional requirement that p can be established by executing transition t with precondition $(we.p)_t$. Specifically, $(we.p)_t$ is the weakest solution of (1) in q; here, $wp.s.q$ is the weakest precondition of s with postcondition q.

$$q \equiv (\langle \forall\ s\ :\ wp.s.q \rangle \wedge wp.t.p) \vee p \qquad (1)$$

Now, *we* can be defined. Below, t is quantified over all transitions.

$$we.p \;\equiv\; \langle \exists\ t \quad :: \quad (we.p)_t \rangle \tag{2}$$

Finally, we define $wlt.q$ as the strongest solution in p of (3).

$$p \;\equiv\; q \;\vee\; we.p \tag{3}$$

The existence of the weakest solution for (1) and the strongest solution for (3) can be established by appealing to the Knaster-Tarski Theorem; see [10] for details. The definition of *wlt* given by equation (3) can be shown to be the same as the definition (W1) of Section 4.6.1. The fixpoint characterization yields a number of properties of *we* and *wlt* (we have seen some of these earlier):

$p \;\Rightarrow\; we.p$
$wlt.p \;\equiv\; p \;\vee\; we.(wlt.p)$
$[(p \;\vee\; we.q) \;\Rightarrow\; q] \;\Rightarrow\; [wlt.p \;\Rightarrow\; q]$
$we.(wlt.p) \;\Rightarrow\; wlt.p$
$p \;\Rightarrow\; wlt.p$
$(p \;\Rightarrow\; q) \;\Rightarrow\; (wlt.p \;\Rightarrow\; wlt.q)$
$wlt.false \;\equiv\; false$
$wlt.(wlt.p) \;\equiv\; wlt.p$
$wlt.(p \;\vee\; wlt.q) \;\Rightarrow\; wlt(p \;\vee\; q)$

4.6.3 The Role of the Disjunction Rule

We have defined $\mapsto$ using three inference rules. The basis and the transitivity rules are intuitively acceptable. The need for the disjunction rule, however, is not easy to see. In this section, we show that the disjunction rule is (1) unnecessary for performing finite disjunctions and (2) necessary for performing infinite disjunctions. One consequence of this observation is that for a finite-state program—where the number of predicates themselves is finite and, hence, disjunction can be performed only over a finite number of predicates—the disjunction rule is unnecessary.

Let $\models\!\!\Rightarrow$ (a "poor cousin" of $\mapsto$) be the transitive closure of *ensures*, i.e., $\models\!\!\Rightarrow$ is defined by

$$\frac{p \;\mathbf{en}\; q}{p \;\models\!\!\Rightarrow\; q}$$

and

$$\frac{p \;\models\!\!\Rightarrow\; q \;,\; q \;\models\!\!\Rightarrow\; r}{p \;\models\!\!\Rightarrow\; r}$$

We show that $\models\!\!\Rightarrow$ is finitely disjunctive, i.e.,

$$\frac{p \;\models\!\!\Rightarrow\; q \;,\; p' \;\models\!\!\Rightarrow\; q}{p \;\vee\; p' \;\models\!\!\Rightarrow\; q}$$

For this proof, we first show

$$\frac{p \;\models\!\!\Rightarrow\; q}{p \;\vee\; r \;\models\!\!\Rightarrow\; q \;\vee\; r} \tag{1}$$

The demonstration uses induction on the structure of the proof of $p \;\models\!\!\Rightarrow\; q$.

(basis) Assume p **en** q
$p \;\vee\; r$ **en** $q \;\vee\; r$, see Exercise (3d)
$p \;\vee\; r \;\models\!\!\Rightarrow\; q \;\vee\; r$, definition of $\models\!\!\Rightarrow$

(transitivity) Assume for some b that $p \;\models\!\!\Rightarrow\; b \;\models\!\!\Rightarrow\; q$
$p \;\vee\; r \;\models\!\!\Rightarrow\; b \;\vee\; r$, induction hypothesis
$b \;\vee\; r \;\models\!\!\Rightarrow\; q \;\vee\; r$, induction hypothesis
$p \;\vee\; r \;\models\!\!\Rightarrow\; q \;\vee\; r$, transitivity

Now, we can prove the finite disjunctivity of $\models\!\!\Rightarrow$.
$p \;\vee\; p' \;\models\!\!\Rightarrow\; q \;\vee\; p'$, from the premise $p \;\models\!\!\Rightarrow\; q$ and (1) using p' as r
$q \;\vee\; p' \;\models\!\!\Rightarrow\; q$, from the premise $p' \;\models\!\!\Rightarrow\; q$ and (1) using q as r
$p \;\vee\; p' \;\models\!\!\Rightarrow\; q$, transitivity on the above two

This concludes the proof that $\models\!\!\Rightarrow$ is finitely disjunctive. Now, we show that $\models\!\!\Rightarrow$ is *not* infinitely disjunctive.

We show that whenever $p \;\models\!\!\Rightarrow\; q$ can be proven for a program, there is some natural number, k, depending on p, q, such that for any state satifying p there is an execution of length at most k that establishes q. If $p \;\models\!\!\Rightarrow\; q$ is proven by p **en** q, then there is a transition t such that

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

For a state satisfying $p \;\wedge\; q$, an empty execution sequence establishes q and for a state satisfying $p \;\wedge\; \neg q$, the sequence consisting of transition t establishes q. Thus, the bound k equals 1, in this case. If $p \;\models\!\!\Rightarrow\; q$ is proven by $p \;\Rightarrow\; r \;\models\!\!\Rightarrow\; q$, then $k_1 + k_2$ is the required bound where k_1 is the bound for $p \;\models\!\!\Rightarrow\; r$ and k_2 for $r \;\models\!\!\Rightarrow\; q$.

Our impossibility result is derived by considering a program that consists of a single statement; the statement decrements by 1 the value of an integer variable, x. It is straightforward to show that in this program

$$\mathit{true} \;\mapsto\; x < 0$$

holds. However,

$$\mathit{true} \;\models\!\!\Rightarrow\; x < 0$$

cannot be proven. Because if it can be proven, there is a bound k associated with this proof. For any state—in particular, $x = k$—we can find an execution sequence of length at most k establishing $x < 0$, which is impossible.

4.7 Synopsis

The major theme of this chapter is a definition of *leads-to* and the promulgation of its manipulation rules. The definition used the auxiliary concept of *transient predicate* (which is used to define *ensures*, that forms the basis for the definition of *leads-to*). Transient predicates are defined directly from the program text, for different forms of fairness.

The manipulation rules for *leads-to* consist of about four lightweight and four heavyweight rules. The prominent ones are cancellation, PSP and induction rules (in addition to transitivity and disjunction rules that appear in the definition of *leads-to*). The examples illustrate how these rules can be effectively applied in practice.

Bibliographic Notes

The treatment of progress properties in this chapter closely follows the original development described in [3]. The only new element is the introduction of transient predicates and the replacement of "*unless*" by "**co**" in the derived rules, such as the PSP and the completion rules. The minimal progress condition is due to Dijkstra[6]. For a comprehensive treatment of fairness see Francez[8] or Manna and Pnueli[18]. Transient predicates have been used informally before (they are called nonquiescent in ([19], [21]); the present definition is inspired by Cohen[4]. The definition of *ensures* for minimal progress appears in Jutla and Rao[11]. The definition of transient predicate under weak fairness is inspired by Lehmann, Pnueli and Stavi's notion[17] of "helpful transitions." The *leads-to* operator was introduced in Lamport[15]; its interpretation in linear temporal logic is in Owicki and Lamport[22]. Our definition, using inference rules, has facilitated proofs of the derived rules using structural induction. Lamport[16] prescribes deducing the progress properties from the conjunction of the fairness assumption—expressed as a formula in temporal logic—and the safety properties. The graph problem (Section 4.5.5 is from [20]; the original inspiration for this problem is from Chandy and Misra[2].

Recently, Chandy has proposed combining progress with stability; he writes $p \hookrightarrow q$ to denote that once p holds in the program, q will eventually hold and will continue to hold thereafter. The operator $\hookrightarrow$ has many pleasing properties including lhs strengthening, rhs weakening, infinite disjunction and transitivity. It is particularly interesting that $\hookrightarrow$ is finitely conjunctive.

For other approaches to verifications of concurrent programs, see[9].

The notion of *wlt* appears in Knapp[13] and in Jutla, Knapp and Rao[10]. The latter paper includes the fixpoint characterization of *wlt* given in Section 4.6.2. The property (W5) in Section 4.6.1 is due to Singh[26]. Several

varieties of completeness (and incompleteness) results have been established for *leads-to*. Jutla and Rao[11] contains an excellent exposition of what completeness means in this context. They argue that relative completeness in the sense of Cook[5] is all that we can hope for. Such completeness results appear in Cohen[4], Jutla and Rao[11], Knapp[14], Pachl[23], and Rao ([24] and [25]). The fact that the disjunction rule, in the definition of *leads-to*, is unnecessary for finite-state programs is due to van de Snepscheut[27]. It has been exploited by Kaltenbach[12] in implementing an automatic verifier for finite state programs. The *leads-to* operator has been extended to probabilistic programs in [25]. Carruth[1] shows how the progress operators can be extended for stating and proving real-time progress properties. His extensions preserve the derived rules stated in this chapter.

4.8 Exercises

1. Show that the following property holds for the program **Fairness** of Section 4.2, under both minimal progress and weak fairness assumptions. For any integer k

$$x = k \ \wedge \ y = k \ \ \textbf{en} \ \ x + y > 2k$$

2. (Simulating minimal progress by weak fairness) A program consisting of the following statements is executed under the minimal progress condition.

$$x \ := \ 0 \ \ [\!] \ \ x \ := \ 1$$

   Construct a program that operates under weak fairness and "simulates" the above program. Specifically, there should be a 1–1 correspondence between the execution sequences of the above program and the simulator, where two execution sequences correspond if they assign the same sequence of values to x.

3. Show the following for *ensures*.

   (a) (implication) $$\frac{p \ \Rightarrow \ q}{p \ \textbf{en} \ q}$$

   (b) (rhs weakening) $$\frac{p \ \textbf{en} \ q}{p \ \textbf{en} \ q \ \vee \ r}$$

   (c) (lhs manipulation) $$p \ \textbf{en} \ q \ \equiv \ p' \ \textbf{en} \ q$$
   where $p \ \wedge \ \neg q \ \Rightarrow \ p' \ \Rightarrow \ p \ \vee \ q$

Applying this rule show that the following properties are equivalent.

$$p \ \mathbf{en}\ q \ \vee \ r \ , \ p \ \vee \ r \ \mathbf{en}\ q \ \vee \ r \ , \ p \ \wedge \ \neg q \ \mathbf{en}\ q \ \vee \ r$$

(d)
$$\frac{p \ \mathbf{en}\ q}{p \ \vee \ r \ \mathbf{en}\ q \ \vee \ r}$$

(e)
$$\frac{p \ \vee \ q \ \mathbf{en}\ r}{p \ \mathbf{en}\ q \ \vee \ r}$$

(f) (PSP)
$$\frac{\begin{array}{c} p \ \mathbf{en}\ q \\ r \ \mathbf{co}\ s \end{array}}{p \ \wedge \ r \ \mathbf{en}\ (q \ \wedge \ s) \ \vee \ (p \ \wedge \ \neg r \ \wedge \ s)}$$

(g)
$$p \text{ transient } \equiv \ \mathit{true}\ \mathbf{en}\ \neg p$$

4. The following alternative definition for *leads-to* replaces the original basis rule by the two rules shown below.

(implication)
$$\frac{p \ \Rightarrow \ q}{p \ \mapsto \ q}$$

(basis)
$$\frac{p \ \mathbf{co}\ p \ \vee \ q \ , \ p \text{ transient}}{p \ \mapsto \ q}$$

The transitivity and the disjunction rules are retained. Prove that this definition of *leads-to* is equivalent to the original definition.

5. Consider the program consisting of the statements

$$\begin{array}{llll} & x \ := \ x+1 & \text{if} & b \\ [\!] & x \ := \ x-1 & \text{if} & \neg b \end{array}$$

where x is integer and b is boolean. Show that under weak fairness

$$\mathit{true} \ \mapsto \ |x| > m$$

for any integer m.

Can this property be deduced if the program had an additional statement

$$b \ := \ \neg b$$

6. In the following program show that $true \mapsto x, y = 0, 0$ under weak-fairness

Program F
declare x, y, z : natural
assign
$x, y := x - 1, z$ if $x > 0$
$[]\ y := y - 1$ if $y > 0$
$[]\ z := z + 1$
end F

Can you place a bound, as a function of the initial values of x, y, on the number of assignments to x, y required to establish $x, y = 0, 0$?

7. (Working with the lightweight rules) Show for arbitrary predicates p, q, r, p', q'

(a) $$p \lor q \mapsto r \Rightarrow p \mapsto r$$

(b) $$p \land q \mapsto r \Rightarrow p \mapsto \neg q \lor r$$

(c) $$\frac{p \mapsto q \ , \ p' \mapsto q'}{p \lor p' \mapsto (q \land \neg p') \lor q'}$$

(d) Show that for any r where $p \land \neg q \Rightarrow r \Rightarrow p \lor q$,

$$p \mapsto q \equiv r \mapsto q$$

In particular,

$$p \mapsto q \equiv p \land \neg q \mapsto q$$
$$p \mapsto q \equiv p \lor q \mapsto q$$

(e) $$\langle \forall\ i \ :: \ p.i \mapsto q \rangle \equiv \langle \exists\ i \ :: \ p.i \rangle \mapsto q$$

8. (Working with the heavyweight rules) Show for arbitrary predicates p, q, r, b

(a) $$\frac{p \mapsto q \ , \ r \land \neg q \quad \text{stable}}{p \land r \Rightarrow q}$$

(b) $$\frac{p \mapsto q}{FP \Rightarrow (p \Rightarrow q)} \quad ,$$ where FP is the fixedpoint predicate

(c) $\dfrac{p \mapsto \neg p \ , \ \neg p \mapsto p}{\neg FP}$

9. Show that boolean variable b is infinitely-often *true* in the following program under the weak fairness assumption. Here x, y are integers.

$$\begin{array}{lllll} & x & := & x+1 & \text{if} \quad b \\ [\!] & y & := & y+1 & \text{if} \quad \neg b \\ [\!] & b & := & \neg b & \text{if} \quad x \neq y \end{array}$$

10. (Simulating strong fairness by weak fairness) The strong-semaphore program (Section 4.5.6) is reproduced below.

$$\begin{array}{lllllll} & \alpha & :: & s, x & := & \mathit{false}, x+1 & \text{if} \quad s \\ [\!] & \beta & :: & s, y & := & \mathit{false}, y+1 & \text{if} \quad s \\ [\!] & \gamma & :: & s & := & \mathit{true} & \end{array}$$

Modify this program so that under weak fairness we can guarantee: x will be increased infinitely often. There should be a 1–1 correspondence between the execution sequences of the program given above in which x is increased infinitely often and the fair execution sequences of the simulating program, so that (s, x, y) are assigned the same sequence of values in the corresponding sequences.

11. Given that integer variables x, y do not change synchronously and that at least one of them changes eventually, show that x differs from y infinitely often.

12. Show counterexamples to the following.

(a) $\dfrac{p \ \mathbf{en} \ q \ , \ p' \ \mathbf{en} \ q'}{p \ \vee \ p' \ \mathbf{en} \ q \ \vee \ q'}$

(b) $\dfrac{p \ \wedge \ \neg q \ \mathbf{co} \ p \ \vee \ q \ , \ p \mapsto q}{p \ \mathbf{en} \ q}$

(c) $\dfrac{p \mapsto q \ , \ p' \mapsto q'}{p \ \wedge \ p' \mapsto q \ \wedge \ q'}$

(d) $\dfrac{p \mapsto q \ , \ q \mapsto p}{p \equiv q}$

13. (Methodology in applying PSP)

(a) The safety property, "x is nondecreasing," can be written in the following equivalent forms. Using free variable m,

$$x \geq m \text{ stable}$$
$$\text{or, } x = m \ \mathbf{co} \ x \geq m$$

Which of these forms is preferable for applying the PSP rule?

(b) Given p **co** q , p' **co** q' and $r \mapsto b$, compute predicate s in the following where s is as strong as possible.

$$(p \wedge p') \wedge r \mapsto s$$

Does this suggest a methodology for applying the PSP rule?

14. (Induction)

(a) Show

$$\frac{x = k \mapsto x = k - 1}{x > 0 \mapsto x = 0}$$

(b) Let $A[0..N]$ be an array of integers. Define M to be the number of *inversions* in A, i.e.,

$$M = \langle + \, i, j \; : \; 0 \leq i < j \leq N \wedge A[i] > A[j] \; : \; 1 \rangle$$

Show that for $m > 0$:

$$\frac{M = m \mapsto M < m}{true \mapsto \langle \forall \, i \; : \; 0 \leq i < N \; : \; A[i] \leq A[i+1] \rangle}$$

(c) If x decreases eventually and y never decreases then show that $x < y$ holds infinitely often. Is $x < y$ stable?

(d) Items are added to a set s only if p holds as a precondition. If s is nonempty some item is eventually removed from s. Show that eventually either s is empty or p holds.

15. (Completion) An idle process can become active by acquiring a set of resources. The rules for state transition and resource acquisition are the following. Let R be a fixed set of resources.

(a) If the process is idle and holds all resources in R then it becomes active, eventually.

(b) The process does not release any resource r, $r \in R$, as long as it is idle.

(c) For any r, $r \in R$, the process eventually holds r if it remains idle.

Show that the process does not stay idle forever. Note that an idle process may become active without holding all resources in R; the stated properties do not prevent this possibility.

16. Show that the completion rules does not hold for an infinite set of predicate pairs (p_i, q_i).

17. (Generalizations of the completion rule)

(a) Prove, for $0 \leq i < N$ and any set S of indices between 0 and $N-1$,

$$\frac{\langle \forall\ i\ :\ 0 \leq i < N\ :\ p_i\ \mapsto\ q_i\ ,\ q_i \quad \text{stable} \rangle}{\begin{array}{l}\langle \forall\ j\ :\ j \in S\ :\ p_j \rangle\ \mapsto \\ \qquad \langle \forall\ j\ :\ j \in S\ :\ q_j \rangle\ \wedge\ \langle \forall\ j\ :\ j \notin S\ :\ p_j\ \Rightarrow\ q_j \rangle\end{array}}$$

(b) Prove, for $0 \leq i < N$

$$\frac{\begin{array}{c} p_i\ \mapsto\ q_i\ \vee\ r_i \\ q_i\ \mathbf{co}\ q_i\ \vee\ r_i \end{array}}{\langle \forall\ i\ ::\ p_i \rangle\ \mapsto\ \langle \forall\ i\ ::\ q_i \rangle\ \vee\ \langle \exists\ i\ ::\ r_i \rangle}$$

Hint: Let $b \triangleq \langle \exists\ i\ ::\ r_i \rangle$. Then, by weakening the rhs of the premises,

$$\begin{array}{l} p_i\ \mapsto\ q_i\ \vee\ b \\ q_i\ \mathbf{co}\ q_i\ \vee\ b \end{array}$$

Show, from these premises, that

$$\langle \forall\ i\ ::\ p_i \rangle\ \mapsto\ \langle \forall\ i\ ::\ q_i \rangle\ \vee\ b$$

This is the completion rule, stated in Section 4.4.4.

(c) Propose and prove a general version combining Exercises (17a) and (17b).

18. Prove that the transitivity and the disjunction rules in the definition of $\mapsto$ can be combined into the following single rule. For any set S of predicates,

$$\frac{p\ \mapsto\ \langle \exists\ q\ :\ q \in S\ :\ q \rangle\ ,\ \langle \forall\ q\ :\ q \in S\ :\ q\ \mapsto\ r \rangle}{p\ \mapsto\ r}$$

19. Show (using W1–W5 of Section 4.6.1)

$$q \text{ stable}\ \Rightarrow\ \mathit{wlt}.q \text{ stable}$$

20. Show

$$\frac{p\ \mapsto\ q\ ,\ \neg r\ \wedge\ \neg q \quad \text{stable}}{\begin{array}{c} p\ \Rightarrow\ q\ \vee\ r, \\ p\ \wedge\ \neg q\ \mathbf{co}\ q\ \vee\ r \end{array}}$$

Hint: Use the properties of *wlt* (Section 4.6.1).

4.9 Solutions to Exercises

1. (minimal progress) The proof obligations are

$$\begin{array}{l} x = k \ \wedge \ y = k \ \Rightarrow \ \mathit{true} \qquad \text{and,} \\ \{x = k \ \wedge \ y = k\} \ \ x \ := \ x + 1 \ \ \{x + y > 2k\} \\ \{x = k \ \wedge \ y = k\} \ \ y \ := \ y + 1 \ \ \{x + y > 2k\} \\ \{x = k \ \wedge \ y = k \ \wedge \ x \neq y\} \ \ z \ := \ z + 1 \ \ \{x + y > 2k\} \end{array}$$

   These are easily established (the last assertion is of the form $\{\mathit{false}\}\ s\ \{p\}$).
   (weak fairness) The proof obligations are

$$x = k \ \wedge \ y = k \ \mathbf{co} \ (x = k \ \wedge \ y = k) \ \vee \ (x + y > 2k)$$

   , straightforward

   and, $\langle \exists \ s \ :: \ \{x = k \ \wedge \ y = k\} \ \ s \ \ \{x + y > 2k\} \rangle$

   For the last demonstration, use the statement that increments x (or y).

2. Introduce a boolean variable b. The program consists of

$$\begin{array}{l} x \ := \ 0 \quad \text{if} \quad b \qquad \sim \qquad 1 \quad \text{if} \quad \neg b \\ [\!] \ \ b \ := \ \neg b \end{array}$$

3. (a) See proof of the implication rule for $\mapsto$ (Section 4.4.4)

   (b)

	$p \ \wedge \ \neg q \ \mathbf{co} \ p \ \vee \ q$	, from $p \ \mathbf{en} \ q$
(1)	$p \ \wedge \ \neg q \ \wedge \ \neg r \ \mathbf{co} \ p \ \vee \ q \ \vee \ r$	, strengthening lhs and weakening the rhs
	$p \ \wedge \ \neg q$ transient	, from $p \ \mathbf{en} \ q$
	$p \ \wedge \ \neg q \ \wedge \ \neg r$ transient	, strengthening
	$p \ \mathbf{en} \ q \ \vee \ r$	, from (1) and above

   (c) From $p \ \wedge \ \neg q \ \Rightarrow \ p' \ \Rightarrow \ p \ \vee \ q$,

$$p' \ \wedge \ \neg q \ \equiv \ p \ \wedge \ \neg q \text{ and } p' \ \vee \ q \ \equiv \ p \ \vee \ q.$$

   Replacing $p \ \wedge \ \neg q$, $p \ \vee \ q$ in the definition of $p \ \mathbf{en} \ q$ by $p' \ \wedge \ \neg q$, $p' \ \vee \ q$, respectively, the result follows. The given equivalences can be proven easily using this rule.

   (d)

$p \ \mathbf{en} \ q \ \vee \ r$	, weakening rhs of the premise (Exercise (3b))
$p \ \vee \ r \ \mathbf{en} \ q \ \vee \ r$	, from Exercise (3c)

   (e)

$p \ \vee \ q \ \mathbf{en} \ q \ \vee \ r$	, weakening rhs of the premise (Exercise (3b))
$p \ \mathbf{en} \ q \ \vee \ r$	, from Exercise (3c), interchanging q, r

(f) See proof of the PSP rule for $\mapsto$, basis case (Section 4.4.4).

(g)
$$\begin{array}{ll} & \mathit{true}\ \mathbf{en}\ \neg p \\ \equiv & \{\text{definition of } \mathbf{en}\} \\ & p\ \mathbf{co}\ \mathit{true}\ \wedge\ p \text{ transient} \\ \equiv & \{p\ \mathbf{co}\ \mathit{true}, \text{ holds for all } p\} \\ & p \text{ transient} \end{array}$$

4. Proof is by mutual implication.

   - original definition $\Rightarrow$ proposed definition:
     The implication rule has been proven in Section 4.4.4. The transitivity and disjunction rules follow trivially. So it remains to show, under the original definition of $\mapsto$,

$$\frac{p\ \mathbf{co}\ p\ \vee\ q\ ,\ p \text{ transient}}{p\ \mapsto\ q}$$

     $p\ \wedge\ \neg q\ \mathbf{co}\ p\ \vee\ q$, strengthening lhs of the **co**-property in the premise
     $p\ \wedge\ \neg q$ transient , strengthening the premise
     $p\ \mathbf{en}\ q$, definition of **en**
     $p\ \mapsto\ q$, definition of $\mapsto$

   - proposed definition $\Rightarrow$ original definition:
     We prove the basis rule of the original definition under the new interpretation for $\mapsto$ (the other two rules have been retained).

$$\frac{p\ \mathbf{en}\ q}{p\ \mapsto\ q}$$

     $p\ \wedge\ \neg q\ \mathbf{co}\ p\ \vee\ q$, from $p\ \mathbf{en}\ q$ in the premise
     $p\ \wedge\ \neg q$ transient , from $p\ \mathbf{en}\ q$ in the premise
     $p\ \wedge\ \neg q\ \mapsto\ q$, from the basis rule of the proposed definition (using $p\ \wedge\ \neg q$ for p)
     $p\ \wedge\ q\ \mapsto\ q$, from the implication rule in the proposed definition
     $p\ \mapsto\ q$, disjunction rule of the proposed definition

5. For any integer k verify that

   $x = k\ \wedge\ b\ \mathbf{en}\ x > k\ \wedge\ b$ (using the statement that increments x, as the helpful transition)

   Then,

$$\begin{array}{ll} x = k\ \wedge\ b\ \mapsto\ x > k\ \wedge\ b & \text{, from } x = k\ \wedge\ b\ \mathbf{en}\ x > k\ \wedge\ b \\ b\ \mapsto\ x > m & \text{, induction on integers} \\ b\ \mapsto\ |x| > m & \text{, weakening rhs} \qquad (1) \\ \neg b\ \mapsto\ x < -m & \text{, similarly} \end{array}$$

$\neg b \ \mapsto \ \|x\| > m$	, weakening rhs ($x < -m \ \Rightarrow -x > m$)
$true \ \mapsto \ \|x\| > m$	, disjunction of above and (1)

This property cannot be deduced if $b \ := \ \neg b$ is an additional statement in the program because starting in a state where $\neg b$ holds the following execution never changes x.

loop
$\quad x \ := \ x + 1 \quad$ if $\quad b$
; $\ b \ := \ \neg b$
; $\ x \ := \ x - 1 \quad$ if $\quad \neg b$
; $\ b \ := \ \neg b$
forever

6. For natural numbers m, n define $(x, y) \prec (m, n)$ to mean that (x, y) is lexicographically smaller than (m, n), i.e., $x < m \vee (x = m \wedge y < n)$. Show from the text of F that

$$x, y = m, n \ \wedge \ m \neq 0 \ \textbf{en} \quad (x, y) \prec (m, n)$$
and, $$x, y = m, n \ \wedge \ n \neq 0 \ \textbf{en} \quad (x, y) \prec (m, n)$$

Converting the above two properties to *leads-to*s and taking their disjunction

$$x, y = m, n \ \wedge \ m, n \neq 0, 0 \ \mapsto \ (x, y) \prec (m, n)$$

Applying corollary of induction (with $p, q, M, m, r.m$ as *true*, *false*, (x, y), (m, n), $(m, n \neq 0, 0)$, respectively), and noting that $\prec$ is a well-founded order over pairs of naturals,

$$true \ \mapsto \ x, y = 0, 0$$

There is no bound on the number of assignments required to establish $x, y = 0, 0$. This is because z could be made arbitrarily large, and hence, y could be made arbitrarily large with every decrease in the value of x.

7. (a) strengthen lhs of the premise to p

(b)

$p \ \wedge \ q \ \mapsto \ r$	, premise
$p \ \wedge \ \neg q \ \mapsto \ \neg q$	, implication
$p \ \mapsto \ \neg q \ \vee \ r$	, disjunction

(c)
$p \mapsto q$, premise
$p \mapsto (q \wedge \neg p') \vee p'$, weakening the rhs
$p' \mapsto q'$, premise
$p \mapsto (q \wedge \neg p') \vee q'$, cancellation on the above two
$p \vee p' \mapsto (q \wedge \neg p') \vee q'$
, disjunction on the above two

(d) Proof is by mutual derivation.

Proof of $(p \mapsto q) \Rightarrow (r \mapsto q)$:
$p \mapsto q$, premise
$q \mapsto q$, implication
$p \vee q \mapsto q$, disjunction
$r \mapsto q$, strengthening lhs (use $r \Rightarrow p \vee q$)

Proof of $(r \mapsto q) \Rightarrow (p \mapsto q)$:
From $p \wedge \neg q \Rightarrow r \Rightarrow p \vee q$, we deduce
$r \wedge \neg q \Rightarrow p \Rightarrow r \vee q$

The result follows from the first proof, interchanging the roles of p, r.

(e)
$\langle \forall\ i \ :: \ p_i \mapsto q \rangle$
$\Rightarrow$ {disjunction} $\langle \exists\ i \ :: \ p_i \rangle \mapsto q$
Conversely,
$\langle \exists\ i \ :: \ p_i \rangle \mapsto q$, premise
$\langle \forall\ j \ :: \ p_j \mapsto q \rangle$, strengthening the lhs

8. (a)
$p \mapsto q$, premise
$r \wedge \neg q$ stable , premise
$p \wedge r \wedge \neg q \mapsto \mathit{false}$
, stable conjunction
$\neg(p \wedge r \wedge \neg q)$, impossibility
$p \wedge r \Rightarrow q$, simplifying

(b)
$p \mapsto q$, premise
$FP \wedge \neg q$ stable , stability at fixed point
$FP \Rightarrow (p \Rightarrow q)$, from Exercise (8a) using FP for r

(c)
$FP \Rightarrow (p \Rightarrow \neg p)$, from Exercise (8b) using $p \mapsto \neg p$
$FP \Rightarrow \neg p$, simplifying
$FP \Rightarrow p$, similarly, from $\neg p \mapsto p$
$FP \Rightarrow \mathit{false}$, conjunction of the above two
$\neg FP$, predicate calculus

9. Prove the following as *ensures* properties from the program text.

1. $x < y \mapsto b$

2. $x = y \;\mapsto\; x < y \;\vee\; b$
3. $x > y \;\mapsto\; x \leq y \;\vee\; b$

Now,

$x = y \;\mapsto\; b$	, cancel in rhs of (2) using (1)
$x \leq y \;\mapsto\; b$	, disjunction of above and (1)
$x > y \;\mapsto\; b$	, cancel in rhs of (3) using above
$true \;\mapsto\; b$	, disjunction of the above three

10. Introduce a boolean variable b. Variable b becomes *true* and remains *true* until x increases. Furthermore, x is guaranteed to increase once b becomes *true*.

initially $b = false$
assign

α'	:: $s, x, b \;:=\; false, x+1, false$	if	s
$[\!]\ \beta'$	:: $s, y \;:=\; false, y+1$	if	$s \;\wedge\; \neg b$
$[\!]\ \gamma'$	:: $s \;:=\; true$		
$[\!]\ \delta'$	:: $b \;:=\; true$		

We show that x increases beyond any bound. Prove the following directly from the program text. All the *leads-to* properties are *ensures* (under weak fairness). For any integer k

1. $true \;\mapsto\; b$
2. $b \;\mapsto\; b \;\wedge\; s$
3. $b \;\wedge\; s \;\wedge\; x \geq k \;\mapsto\; x > k$
4. $x \geq k$ stable

Now,

$true \;\mapsto\; b \;\wedge\; s$	, transitivity on (1,2)
$x \geq k \;\mapsto\; b \;\wedge\; s \;\wedge\; x \geq k$	, stable conjunction of above and (4)
$x \geq k \;\mapsto\; x > k$	, transitivity on the above and (3)
$x = k \;\mapsto\; x > k$	, strengthen lhs
$\langle \forall\ m \;::\; true \;\mapsto\; x > m \rangle$	, induction on integers

To establish the 1–1 correspondence, for any execution sequence, E, of the original program in which x is increased infinitely often, construct an execution sequence, E', of the simulating program by replacing each α, β, γ operation in E by $\gamma'\beta'\delta'\alpha', \beta', \gamma'$, respectively. We leave it to the reader to show that (s, x, y) will be assigned the same sequence of values in both E and E'. Further, since x is increased infinitely often in E, sequence E contains an infinite number of α operations; therefore, E' contains an infinite number of $\alpha', \beta', \gamma', \delta'$ operations (because α is replaced by $\gamma'\beta'\delta'\alpha'$); hence, E' is fair. The remaining proof obligation—to show that for every fair execution sequence of the simulating program there exists a corresponding execution sequence of the original program—is left to the reader.

11. x, y do not change synchronously. For any integers m, n

    1. $x, y = m, n$ **co** $x = m \ \vee \ y = n$

    At least one of x, y changes eventually.

    2. $x, y = m, n \ \mapsto \ x \neq m \ \vee \ y \neq n$

    We have

    $true \ \mapsto \ x \neq y$

    Proof:

    $x, y = m, n \ \mapsto \ (x = m \ \wedge \ y \neq n) \ \vee \ (x \neq m \ \wedge \ y = n)$
    , PSP on (1,2)
    $x, y = m, m \ \mapsto \ (x = m \ \wedge \ y \neq m) \ \vee \ (x \neq m \ \wedge \ y = m)$
    , replace n by m
    $x, y = m, m \ \mapsto \ x \neq y$, weakening rhs
    $x = y \ \mapsto \ x \neq y$, disjunction over all m
    $true \ \mapsto \ x \neq y$, disjunction with $x \neq y \ \mapsto \ x \neq y$

12. (a) This solution is due to Roland Backhouse. Consider a program consisting of (with integer variable x)

    $x \ := \ 0$ if $x > 0$
    $[\!]$ $x \ := \ 0$ if $x < 0$

    We can show

    $x > 0$ **en** $x = 0$ and,
    $x < 0$ **en** $x = 0$

    But, we can't show

    $x \neq 0$ **en** $x = 0$

    because there is no statement t such that

    $\{x \neq 0\} \ \ t \ \ \{x = 0\}$

    (b) In the program of Exercise (12a) with

    $p \ \equiv \ x \neq 0 \ , \ q \ \equiv \ x = 0$

    We have

    $p \ \wedge \ \neg q$ **co** $p \ \vee \ q$ (i.e., $x \neq 0$ **co** $true$)

    and $p \ \mapsto \ q$

    because $x > 0 \ \mapsto \ x = 0$, from $x > 0$ **en** $x = 0$
    $x < 0 \ \mapsto \ x = 0$, from $x < 0$ **en** $x = 0$
    $x \neq 0 \ \mapsto \ x = 0$, disjunction
    $p \ \mapsto \ q$, definition of p, q

    But as we have shown in Exercise (12a) p **en** q can't be proven.

    (c) Consider a program consisting of a single statement (over boolean b).

    initially b
    assign $b \ := \ \neg b$

    We have

    $b \ \mapsto \ \neg b$, from b **en** $\neg b$
    $b \ \mapsto \ b$, implication
    $b \ \mapsto \ false$, applying the alleged inference rule

$\neg b$, impossibility

The conclusion is invalid since it asserts that $\neg b$ is invariant which contradicts that b holds initially.

(d) In the program shown in Exercise (12c) $b \mapsto \neg b$ and $\neg b \mapsto b$. Yet, the conclusion, $b \equiv \neg b$ is invalid.

13. (a) Let us attempt applying the PSP rule using $p \mapsto q$ with each of the given forms. The first form yields

(1) $p \wedge x \geq m \mapsto q \wedge x \geq m$

and, the second form yields

(2) $p \wedge x = m \mapsto (q \wedge x \geq m) \vee x > m$

We can deduce (2) from (1) by strengthening the lhs and weakening the rhs of the latter. We cannot, however, deduce (1) from (2) in general. Therefore, the first form is preferable for doing PSP.

(b) We consider two strategies.

i. Apply PSP twice.

p **co** q , premise
$r \mapsto b$, premise
$p \wedge r \mapsto (q \wedge b) \vee (\neg p \wedge q)$, PSP
p' **co** q' , premise
$p \wedge p' \wedge r \mapsto$
$(q' \wedge q \wedge b) \vee (\neg p \wedge q' \wedge q) \vee (\neg p' \wedge q')$
, PSP

ii. Conjoin the **co**s and then apply PSP

p **co** q , premise
p' **co** q' , premise
$p \wedge p'$ **co** $q \wedge q'$, conjunction
$r \mapsto b$, premise
$p \wedge p' \wedge r \mapsto$
$[(q \wedge q') \wedge b] \vee \neg(p \wedge p') \wedge (q \wedge q')$
, PSP

The rhs of the conclusion in (b-ii) is stronger than the rhs of the conclusion in (b-i). This suggests that safety properties be computed as far as possible before applying PSP.

14. (a)

$x = k \mapsto x = k - 1$, premise
$x = k \wedge k > 0 \mapsto x = k - 1 \wedge k > 0$
, stable conjunction with $k > 0$
$x = k \wedge x > 0 \mapsto (x < k \wedge x > 0) \vee x = 0$
, rewriting lhs and rhs
$x > 0 \mapsto x = 0$, induction over integers (use

the range, $k > 0$)

(b) $\langle \forall\ m\ :\ m > 0\ :\ M = m\ \mapsto\ M < m \rangle$
, premise
$true\ \mapsto\ M \leq 0$, induction on integers

Now, $M \leq 0$
$\equiv \langle +\ i, j\ :\ 0 \leq i < j \leq N\ \wedge\ A[i] > A[j]\ :\ 1 \rangle \leq 0$
$\equiv \langle \forall\ i, j\ :\ 0 \leq i < j \leq N\ \wedge\ A[i] \leq A[j] \rangle$
$\Rightarrow$ {instantiating $j = i + 1$}
$\langle \forall\ i\ :\ 0 \leq i < N\ :\ A[i] \leq A[i+1] \rangle$

(c) For any m, n

$x = m\ \mapsto\ x < m$, premise
$true\ \mapsto\ x < n$, induction on integers
$y \geq n$ stable , premise
$y \geq n\ \mapsto\ x < n\ \wedge\ y \geq n$, stable conjunction
$y = n\ \mapsto\ x < y$, strengthening lhs, weakening rhs
$true\ \mapsto\ x < y$, disjunction over n

To see that $x < y$ is not stable: Even though x is decreased eventually, say from m to n, x may be increased beyond m and (beyond y) between these two points in the computation.

(d) For any arbitrary set S

$\neg p\ \wedge\ s = S$ **co** $s \subseteq S$, premise
$s = S\ \wedge\ s \neq \phi\ \mapsto\ S - s \neq \phi$, premise
$\neg p\ \wedge\ s = S\ \wedge\ s \neq \phi\ \mapsto\ (s \subseteq S\ \wedge\ S - s \neq \phi)$
$\vee\ (p\ \wedge\ s \subseteq S)\ \vee\ s \subset S$, PSP
$\neg p\ \wedge\ s = S\ \wedge\ s \neq \phi\ \mapsto\ s \subset S\ \vee\ p$, weakening the rhs
$p\ \wedge\ s = S\ \wedge\ s \neq \phi\ \mapsto\ p$, implication
$s = S\ \wedge\ s = \phi\ \mapsto\ s = \phi$, implication
$s = S\ \mapsto\ s \subset S\ \vee\ s = \phi\ \vee\ p$, disjunction of the above three
$true\ \mapsto\ s = \phi\ \vee\ p$, induction

15. Use $idle$, $\neg idle$ to denote that the process is idle, active, respectively. Let H denote the set of resources that the process holds. In the following, r ranges over all elements of R.

(a) $idle\ \wedge\ R\ \subseteq\ H\ \mapsto\ \neg idle$
(b) $idle\ \wedge\ r \in H$ **co** $\neg idle\ \vee\ r \in H$
(c) $idle\ \mapsto\ \neg idle\ \vee\ r \in H$

We have to show,
$true\ \mapsto\ \neg idle$

Proof: Observe that the rhs of (b,c) can be written as
$(idle\ \wedge\ r \in H)\ \vee\ \neg idle$.
Applying completion on (b,c) over all r, $r \in R$

$idle \;\mapsto\; \langle \forall\, r \;:\; r \in R \;:\; idle \wedge r \in H \rangle \;\vee\; \neg idle$
$idle \;\mapsto\; (idle \wedge R \subseteq H) \;\vee\; \neg idle$
, simplifying
$idle \;\mapsto\; \neg idle$, cancellation of the above and (a)
$true \;\mapsto\; \neg idle$, disjunction with $\neg idle \;\mapsto\; \neg idle$

16. Consider a program consisting of a single statement that increments the integer variable, x.

$x \;:=\; x+1$
For any integer k, $k \geq 0$,
$x = 0 \;\mapsto\; x > k$ and,
$x > k$ stable
Yet, $\langle \wedge\, k \;::\; x = 0 \rangle \;\mapsto\; \langle \wedge\, k \;::\; x > k \rangle$
which is equivalent to
$x = 0 \;\mapsto\; false$
is, clearly, invalid.

17. (a) The following proof is due to Ernie Cohen.

For k, $0 \leq k \leq N$, let
1. $A_k \;\triangleq\; \langle \forall\, i \;:\; 0 \leq i < k \;:\; p_i \;\Rightarrow\; q_i \rangle$
We show for all k, $0 \leq k \leq N$
2. $true \;\mapsto\; A_k$
Proof of (2) is by induction on k.
$k = 0 \;:\; true \;\mapsto\; A_0$, trivially from $A_0 = true$
$k+1, 0 \leq k < N \;:$
$true$
$\mapsto$ {induction hypothesis}
A_k
$\mapsto$ {weakening}
$(A_k \wedge \neg p_k) \vee p_k$
$\mapsto$ {cancellation using $p_k \;\mapsto\; q_k$}
$(A_k \wedge \neg p_k) \vee q_k$
$\mapsto$ {stable conjunction of $true \;\mapsto\; A_k$ and q_k stable yields $q_k \;\mapsto\; A_k \wedge q_k$. Apply cancellation.}
$(A_k \wedge \neg p_k) \vee (A_k \wedge q_k)$
$\mapsto$ {definition of A_{k+1}}
A_{k+1}

We are now ready to show the desired conclusion. Here i is quantified over $0 \leq i < N$ unless the quantification is explicit.

$true \;\mapsto\; \langle \forall\, i \;::\; p_i \;\Rightarrow\; q_i \rangle$

, from (2), setting $k = N$
$\langle \forall\ i\ :\ i \in S\ :\ q_i \rangle$ stable
, stable conjunction from the premise
$\langle \forall\ i\ :\ i \in S\ :\ q_i \rangle\ \mapsto$
$\quad \langle \forall\ i\ ::\ p_i\ \Rightarrow\ q_i \rangle\ \wedge\ \langle \forall\ i\ :\ i \in S\ :\ q_i \rangle$
, stable conjunction of the above two
$\langle \forall\ i\ :\ i \in S\ :\ p_i \rangle\ \mapsto\ \langle \forall\ i\ :\ i \in S\ :\ q_i \rangle$
, completion rule on the premises
$\langle \forall\ i\ :\ i \in S\ :\ p_i \rangle\ \mapsto\ \langle \forall\ i\ ::\ p_i\ \Rightarrow\ q_i \rangle\ \wedge$
$\qquad \langle \forall\ i\ :\ i \in S\ :\ q_i \rangle$
, transitivity on the above two
$\langle \forall\ i\ :\ i \in S\ :\ p_i \rangle\ \mapsto$
$\quad \langle \forall\ i\ :\ i \notin S\ :\ p_i\ \Rightarrow\ q_i \rangle\ \wedge\ \langle \forall\ i\ :\ i \in S\ :\ q_i \rangle$
, rewriting rhs

(b) We use the same notation and the proof structure as in the proof of the special case of the completion rule (Section 4.4.4). We show the proof for the inductive case. We have, for any i (from the premises and using the properties of *wlt*)

1. $p_i\ \mapsto\ q_i\ \vee\ b$
2. q_i **co** $q_i\ \vee\ b$
3. $r\ \mapsto\ q_N\ \vee\ b$
4. $p_N\ \Rightarrow\ r$
5. $(q_N\ \vee\ b)\ \Rightarrow\ r$
6. $r\ \wedge\ \neg(q_N\ \vee\ b)$ **co** r

The induction hypothesis is
$P\ \mapsto\ Q\ \vee\ b$

We have to show the inductive case, $P\ \wedge\ p_N\ \mapsto\ (Q\ \wedge\ q_N)\ \vee\ b$.

$P\ \wedge\ p_N$
$\Rightarrow$ {4}
$P\ \wedge\ r$
$\mapsto$ $\{q_N$ **co** $q_N\ \vee\ b$, from (3)
$\quad r\ \wedge\ q_N\ \wedge\ \neg b$ **co** $q_N\ \vee\ b$, lhs strengthening
$\quad r\ \wedge\ \neg q_N\ \wedge\ \neg b$ **co** r , from (6)
$\quad r\ \wedge\ \neg b$ **co** $r\ \vee\ q_N\ \vee\ b$, disjunction
$\quad r\ \wedge\ \neg b$ **co** r , $(q_N\ \vee\ b)\ \Rightarrow\ r$ from (5)
$\quad P\ \mapsto\ Q\ \vee\ b$, induction hypothesis
$\quad P\ \wedge\ r\ \wedge\ \neg b\ \mapsto\ (Q\ \wedge\ r)\ \vee\ b$, PSP and weakening the rhs
$\quad P\ \wedge\ r\ \wedge\ b\ \mapsto\ b$, implication
$\quad P\ \wedge\ r\ \mapsto\ (Q\ \wedge\ r)\ \vee\ b$, disjunction
$\}$
$(Q\ \wedge\ r)\ \vee\ b$

$$
\begin{array}{lll}
\mapsto & \{r \mapsto q_N \vee b & \text{, from (3)} \\
& \ Q \textbf{ co } Q \vee b & \text{, conjunction on (2)} \\
& \ Q \wedge r \mapsto (Q \wedge q_N) \vee b & \text{, PSP} \\
& \ (Q \wedge r) \vee b \mapsto (Q \wedge q_N) \vee b & \text{, disjunction with } b \mapsto b \\
& \} & \\
& (Q \wedge q_N) \vee b &
\end{array}
$$

(c)

$$
\frac{\begin{array}{c} p_i \mapsto q_i \vee r_i \\ q_i \textbf{ co } q_i \vee r_i \end{array}}{\begin{array}{l} \langle \forall\ i : i \in S : p_i \rangle \mapsto [\langle \forall\ i : i \in S : q_i \rangle \wedge \\ \qquad \langle \forall\ i : i \notin S : p_i \Rightarrow q_i \rangle] \vee \langle \exists\ i :: r_i \rangle \end{array}}
$$

Proof left to the reader.

18. To show that the two definitions are equivalent it is best to introduce a new symbol, $\leadsto$, for the second operator:

- First, we show that

$$p \leadsto q \Rightarrow p \mapsto q$$

The proof is by induction on the structure of the proof of $p \leadsto q$.

1. $p \leadsto q$ is proven by p **en** q: Then, from the basis rule, $p \mapsto q$.
2. $p \leadsto q$ is proven by displaying a set S of predicates where
$p \leadsto \langle \exists\ r : r \in S : r \rangle$ and $\langle \forall\ r : r \in S : r \leadsto q \rangle$

Now,

$$
\begin{array}{ll}
\langle \forall\ r : r \in S : r \mapsto q \rangle & \text{, induction hypothesis} \\
\langle \exists\ r : r \in S : r \rangle \mapsto q & \text{, disjunction rule for } \mapsto \\
p \mapsto \langle \exists\ r : r \in S : r \rangle & \text{, induction hypothesis} \\
p \mapsto q & \text{, transitivity for } \mapsto \text{ on the above two}
\end{array}
$$

- Next, we show that

$$p \mapsto q \Rightarrow p \leadsto q$$

A similar proof by induction on the structure of the proof of $p \mapsto q$ is straightforward.

19.

$$
\begin{array}{ll}
wlt.q \wedge \neg q \textbf{ co } wlt.q & \text{, (W5 of Section 4.6.1)} \\
q \textbf{ co } q & \text{, premise} \\
wlt.q \vee q \textbf{ co } wlt.q \vee q & \text{, disjunction} \\
wlt.q \text{ stable} & \text{, } q \Rightarrow wlt.q \text{ (W4 of Section 4.6.1)}
\end{array}
$$

20. (a) Proof of $p \;\Rightarrow\; q \;\vee\; r$: Follows from Exercise (8a)

(b) Proof of $p \;\wedge\; \neg q$ **co** $q \;\vee\; r$:

Consider $wlt.q$ of Section 4.6.1

$wlt.q \;\mapsto\; q$	, (W3)
$\neg r \;\wedge\; \neg q$ stable	, premise
1. $wlt.q \;\Rightarrow\; q \;\vee\; r$	, use $wlt.q$ for p in Exercise (19a)
2. $p \;\Rightarrow\; wlt.q$	, from $p \;\mapsto\; q$ and (W2)
$wlt.q \;\wedge\; \neg q$ **co** $wlt.q$	, (W5)
$p \;\wedge\; \neg q$ **co** $wlt.q$	, strengthening the lhs using (2)
$p \;\wedge\; \neg q$ **co** $q \;\vee\; r$	, weakening the rhs using (1)

References

[1] A. Carruth. Real-time UNITY. Technical Report TR94-10, University of Texas at Austin, Austin, Texas, April 1994.

[2] K. M. Chandy and J. Misra. The drinking philosophers problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, 1984.

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

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

[5] S. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal of Computing*, 7(1):70–90, 1978.

[6] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569, 1965.

[7] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 1989.

[8] N. Francez. *Fairness*. Springer-Verlag, New York, 1986.

[9] Nissim Francez. *Program Verification*. Addison-Wesley, 1992.

[10] 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.

[11] Charanjit S. Jutla and Josyula R. Rao. A methodology for designing proof rules for fair parallel programs. *Formal Aspects of Computing*, 9(4):359–378, 1997.

[12] Markus Kaltenbach. *Interactive Verification Exploiting Program Design Knowledge: A Model-Checker for UNITY.* PhD thesis, The University of Texas at Austin, 1996. http://www.cs.utexas.edu/users/markus/diss.html.

[13] E. Knapp. A predicate transformer for progress. *Information Processing Letters*, 33:323–330, 1990.

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

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

[16] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[17] D. Lehman, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In O. Kariv and S. Even, editors, *Proc. 8th ICALP*. LNCS 115, Springer-Verlag, July 1981.

[18] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems.* Springer-Verlag, 1991.

[19] J. Misra. Reasoning about networks of communicating processes. Presented at the Advanced Nato Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, La Colle-Sur-Loup, France, October 1984.

[20] J. Misra. A theorem about dynamic acyclic graphs. Notes on UNITY: 02–88, The Univ. of Texas at Austin, September 1988.

[21] J. Misra. Equational reasoning about nondeterministic processes. *Formal Aspects of Computing*, 2(2):167–195, 1990.

[22] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, July 1982.

[23] J. Pachl. A simple proof of a completeness result for *leads-to* in the UNITY logic. *Information Processing Letters*, 41:35–38, 1992.

[24] J. R. Rao. On a notion of completeness for the *leads-to*. Notes on UNITY: 24–90, July 1991.

[25] Josyula R. Rao. *Extensions of the UNITY Methodology: Compositionality, Fairness and Probability in Parallelism*, volume 908 of *Lecture Notes in Computer Science*. Springer–Verlag, 1995.

[26] A. K. Singh. A theorem relating *leads-to* and *unless*. Notes on UNITY: 04–88, December 1988.

[27] J. van de Snepscheut. Personal communication.