

# Separation and Reduction

Ernie Cohen

Telcordia Technologies  
ernie@research.telcordia.com

**Abstract.** We present some new theorems that equate an iteration to a sequential composition of stronger iterations, and use these theorems to simplify and generalize a number of known techniques for pretending atomicity in concurrent programs.

## 1 Introduction

One way to simplify the analysis of a concurrent program is to pretend that certain complex operations execute atomically. For example, one can sometimes pretend that messages are received as soon as they are sent, or that servers respond immediately to requests. *Reduction* theorems provide formal justification for such pretenses; they typically depend on commutativity arguments to convert an arbitrary execution to an equivalent one in which complex operations run without interruption.

Proofs of reduction theorems (as well as many other theorems in concurrency control) can be simplified by applying general-purpose *separation* theorems that equate an iteration to a sequential composition of stronger iterations. Separation theorems have received little attention, partly because the obvious theorems are not strong enough for serious applications, and partly because they are awkward to state and prove in formalisms that do not support sequential composition of nondeterministically terminating iterations. Nevertheless, the use of separation has allowed us to consistently generalize and simplify existing reduction techniques; the new reduction theorems

- work in arbitrary contexts (e.g., allowing control to start or end inside of a critical section),
- have weaker hypotheses, (e.g., allowing environment actions to move control in and out of critical sections),
- apply to both terminating and nonterminating executions, and
- have shorter, simpler proofs.

## 2 Omega Algebra

An *omega algebra* is an algebraic structure over the operators (in order of increasing precedence) 0 (nullary), 1 (nullary), + (binary infix),  $\cdot$  (binary infix,

usually written as simple juxtaposition),  $\star$  (binary infix, same precedence as  $\cdot$ ),  $*$  (unary suffix), and  $^\omega$  (unary suffix), satisfying the following axioms<sup>1</sup>:

$$\begin{array}{ll}
(x + y) + z = x + (y + z) & x \leq y \Leftrightarrow x + y = y \\
x + y = y + x & \\
x + x = x & x^* = 1 + x + x^* x^* \\
0 + x = x & x y \leq x \Rightarrow x y^* = x \quad (* \text{ ind}) \\
x (y z) = (x y) z & x y \leq y \Rightarrow x^* y = y \quad (* \text{ ind}) \\
0 x = x 0 = 0 & \\
1 x = x 1 = x & x \star y = x^\omega + x^* y \\
x (y + z) = x y + x z & x^\omega = x x^\omega \\
(x + y) z = x z + y z & x \leq y \ x + z \Rightarrow x \leq y \star z \quad (\star \text{ ind})
\end{array}$$

(In parsing formulas,  $\cdot$  and  $\star$  associate to the right; e.g.,  $u v \star x \star y$  parses to  $(u \cdot (v^\omega + v^* \cdot (x^\omega + x^* \cdot y)))$ ). In proofs, we use the hint “(dist)” to indicate application of the distributivity laws, and the hint “(hyp)” to indicate the use of hypotheses.)

These axioms are sound and complete for the usual equational theory of omega-regular expressions. (Completeness holds only for *standard* terms, where the first arguments to  $\cdot$ ,  $^\omega$ , and  $\star$  are regular.) Thus, we make free use, without proof, of familiar equations from the theory of (omega-)regular languages (e.g.,  $x^* x^* = x^*$ ).

It is easy to see that the axioms not mentioning  $^\omega$  or  $\star$  are closed under duality, where the dual of a formula is obtained by reversing the arguments to  $\cdot$ . Moreover, it can be shown that the remaining axioms are a conservative extension, so the dual of any theorem not mentioning  $^\omega$  or  $\star$  is also a theorem. In hints, we indicate the use of the dual of a theorem by priming the hint (as in (10)').

Many of our theorems treat  $*$  iteration and  $\star$  iteration in a uniform way; to facilitate these, the symbol  $\circ$  (binary infix, right associative, same precedence as  $\cdot$  and  $\star$ ) ranges over the functions  $\star$  and  $(\lambda x, y : x^* y)$ . For example the equation  $x \circ x \circ y = x \circ y$  abbreviates the conjunction  $x^* x^* y = x^* y \wedge x \star x \star y = x \star y$ . Several theorems of this type are used in our proofs:

- (1)  $x \circ y = y + x x \circ y$
- (2)  $x^* x \circ y = x \circ y$
- (3)  $x \star x \circ y = x \star y$
- (4)  $(x + y) \circ z = (y^* x) \circ y \circ z$
- (5)  $(x + y) \circ z = y \circ z + y^* x (x + y) \circ z$
- (6)  $x \circ y \circ z \leq (x + y) \circ z$
- (7)  $x \circ x \circ y = x \circ y$
- (8)  $x \circ y = x^* y + x \circ 0$

<sup>1</sup> The axioms not mentioning  $^\omega$  are equivalent to Kozen's axioms for Kleene algebra[5], plus the three axioms for omega terms.

$y$  is a *complement* of  $x$  iff  $x y = 0$  and  $x + y = 1$ . It is easy to show that complements (when they exist) are unique and that complementation is an involution; a *predicate* is an element of the algebra with a complement. In this paper,  $p, q$ , and  $Q$  range over predicates, with complements  $\bar{p}, \bar{q}$ , and  $\bar{Q}$ . It is easy to show that the predicates form a Boolean algebra, with  $+$  as disjunction,  $\cdot$  as conjunction,  $0$  as *false*,  $1$  as *true*, complementation as negation, and  $\leq$  as implication. Common properties of Boolean algebras (e.g.,  $p q = q p$ ) are used silently in proofs, as is the fact

$$x p y = 0 \implies x y = x \bar{p} y$$

Unlike previous axiomatizations of omega-regular languages, the omega algebra axioms support several interesting programming models, where (intuitively)  $0$  is magic,  $1$  is skip,  $+$  is chaotic nondeterministic choice,  $\cdot$  is sequential composition,  $\leq$  is refinement,  $x^*$  is executed by executing  $x$  any finite number of times, and  $x^\omega$  is executed by executing  $x$  an infinite number of times. (This correspondence holds only for standard terms.) The results of this paper are largely motivated by the *relational model*, where terms denote binary relations over a state space,  $0$  is the empty relation,  $1$  is the identity relation,  $\cdot$  is relational composition,  $+$  is union,  $*$  is reflexive-transitive closure,  $\leq$  is subset, and  $x^\omega$  relates an input state  $s$  to an output state if there is an infinite sequence of states starting with  $s$ , with consecutive states related by  $x$ . (Thus,  $x^\omega$  relates an input state to either all states or none, and  $x^\omega = 0$  iff  $x$  is well-founded.) Predicates are identified with the set of states in their domain (i.e., the states from which they can be executed). In examples, we use the relational model exclusively; however, our theorems are equally applicable to trace-based models, where actions can have permanent effects (like output) that cannot be erased by later divergence.

In using omega algebra to reason about a program, we usually work with two terms, one describing the finite executions of the program, the other describing the infinite executions. However, when working with flat iterations (as we do here), we can merge these terms into a single term using  $\circ$  and a continuation variable. For example, to say that the program that executes  $x$  for a while is equivalent to the program that executes  $y$  for a while, we write the equation  $x \circ z = y \circ z$ ; the instantiation  $\circ = *$ ,  $z = 1$  (i.e.,  $x^* = y^*$ ) says that the two programs have the same finite executions, and the instantiation  $\circ = \star$ ,  $z = 0$  (i.e.,  $x^\omega = y^\omega$ ) says that the two programs have the same infinite executions. Whenever possible, we prefer to work directly with  $\circ$  equations, taking care of both finite and infinite executions with a single proof.

### 3 Iteration

A standard theorem in program verification says that a predicate preserved by the body of a loop is preserved by the loop. The *iteration theorem*, below, generalizes the loop invariance theorem by (1) generalizing the invariant ( $x$ ) from predicates to arbitrary terms; (2) allowing the loop body ( $y$ ) to be transformed

(into  $z$ ) by the passage of the invariant<sup>2</sup>; and (3) allowing for exceptional cases ( $w$ ) where the invariant is not preserved:

$$(9) \quad x y \leq z \ x + w \implies x y \circ u \leq z \circ (x u + w y \circ u)$$

We prove the  $*$  and  $\star$  cases separately. Here's the  $*$  part:

$$\begin{array}{lcl} x y^* u & \leq & \{x y^* \leq z^* (x + w y^*) \text{ (below)}\} \\ z^* (x + w y^*) u & = & \{(\text{dist})\} \\ z^* (x u + w y^* u) & & \\ \\ x y^* & \leq & \{x \leq z^* (x + w y^*)\} \\ z^* (x + w y^*) y^* & = & \{(\text{proof below}); (* \text{ ind})\} \\ z^* (x + w y^*) & & \\ \\ z^* (x + w y^*) y & = & \{(\text{dist})\} \\ z^* x y + z^* w y^* y & \leq & \{x y \leq z x + w \text{ (hyp)}\} \\ z^* (z x + w) + z^* w y^* y & \leq & \{(\text{dist}); z^* z \leq z^*; y^* y \leq y^*\} \\ z^* x + z^* w + z^* w y^* & \leq & \{w \leq w y^*; (\text{dist})\} \\ z^* (x + w y^*) & & \end{array}$$

And here's the  $\star$  part:

$$\begin{array}{lcl} x y \star u & = & \{y \star u = u + y y \star u \text{ (1)}\} \\ x (u + y y \star u) & = & \{(\text{dist})\} \\ x u + x y y \star u & \leq & \{x y \leq z x + w \text{ (hyp)}\} \\ x u + (z x + w) y \star u & = & \{(\text{dist})\} \\ z (x y \star u) + (x u + w y \star u) & & \end{array}$$

$$\text{so } x y \star u \leq z \star (x u + w y \star u) \text{ by } (\star \text{ ind})$$

□

The separation theorems of the next section are often used in tandem with the following “loop elimination” lemmas that remove extraneous iterations:

$$(10) \quad x y = 0 \implies x y \circ z = x z$$

$$\begin{array}{lcl} x y \circ z & \leq & \{x y = 0 \text{ (hyp)}, \text{ so } x y \leq 0 x; \text{ (iter)}\} \\ 0 \circ x z & = & \{0 \circ u = u\} \\ x z & & \end{array}$$

□

$$(11) \quad x y = 0 \implies x \circ y = y + x \circ 0$$

$$\begin{array}{lcl} x \circ y & = & \{(8)\} \\ x^* y + x \circ 0 & = & \{x y = 0 \text{ (hyp)}; (10)'\} \\ y + x \circ 0 & & \end{array}$$

□

---

<sup>2</sup> This allows the iteration theorem to subsume forward data refinement and simulation (and backward refinement and simulation for finite executions).

## 4 Separation

In this section, we prove some general theorems for rewriting an iteration to a sequential composition of stronger iterations. They typically have the form

$$\dots \implies (x + y) \circ z = x \circ y \circ z$$

(In proofs, we show only  $(x + y) \circ z \leq x \circ y \circ z$ ; the reverse inequality follows from (6).) We call these “separation theorems” because they allow a heterogeneous iteration of  $x$ ’s and  $y$ ’s to be partitioned into separate  $x$  and  $y$  iterations. In a programming context, we can think of  $x$  and  $y$  as transition relations; the conclusion says that running  $x$  and  $y$  concurrently for a while has the same effect as running  $x$  for a while and then running  $y$  for a while.

$$(12) \quad y \ x \leq x \ y^* \implies (x + y) \circ z = x \circ y \circ z$$

$$\begin{array}{ll} (x + y) \circ z & \leq \{1 \leq y^*; (x + y) \circ z = (y^* \ x) \circ y \circ z \ (4)\} \\ y^* (y^* \ x) \circ y \circ z & \leq \{y^* (y^* \ x) \leq x \ y^* \text{ (below); (iter)}\} \\ x \circ y^* \ y \circ z & = \{y^* \ y \circ z = y \circ z \ (2)\} \\ x \circ y \circ z & \\ \\ y^* \ y^* \ x & = \{y^* \ y^* = y^*\} \\ y^* \ x & \leq \{y \ x \leq x \ y^* \text{ (hyp); (iter)}'\} \\ x \ (y^*)^* & = \{(y^*)^* = y^*\} \\ x \ y^* & \end{array}$$

□

The hypotheses of (12) can be weakened in several ways. The first two variations are based on the following lemma:

$$(13) \quad y \ x \leq x \ (x + y)^* \implies (x + y) \circ z \leq x \star y \circ z$$

$$\begin{array}{ll} (x + y) \circ z & = \{(5)\} \\ y \circ z + y^* \ x \ (x + y) \circ z & \leq \{y \ x \leq x \ (x + y)^* \text{ (hyp); (iter)}'\} \\ y \circ z + x \ ((x + y)^*)^* \ (x + y) \circ z & = \{(u^*)^* = u^*; u^* \ u \circ z = u \circ z \ (2)\} \\ y \circ z + x \ (x + y) \circ z & \end{array}$$

$$\text{so } (x + y) \circ z \leq x \star y \circ z \text{ by } (\star \text{ ind})$$

□

Two separation theorems follow immediately from (13). For  $\star$  separation, we can strengthen the conclusion to an equality, and we can obtain a  $*$  separation theorem by eliminating infinite executions of  $x$ :

$$(14) \quad y \ x \leq x \ (x + y)^* \implies (x + y) \star z = x \star y \star z$$

$$(15) \quad x^\omega = 0 \ \wedge \ y \ x \leq x \ (x + y)^* \implies (x + y)^* = x^* \ y^*$$

The hypothesis  $x^\omega = 0$  of (15) cannot be eliminated. For example, if  $x = \{\langle 0, 1 \rangle, \langle 1, 2 \rangle, \langle 2, 2 \rangle\}$  and  $y = \{\langle 1, 0 \rangle, \langle 2, 0 \rangle\}$ , then  $y x = \{\langle 1, 1 \rangle, \langle 2, 1 \rangle\} \leq x y x$ , and  $\langle 2, 1 \rangle \in y x$ , but  $\langle 2, 1 \rangle \notin x^* y^*$ .

In the special case of  $*$  separation, (12) can be strengthened as follows:

$$(16) \quad y x \leq (x + 1) y^* \implies (x + y)^* = x^* y^*$$

$$\begin{array}{lcl} (x + y)^* & \leq & \{u^* = (u + 1)^* \} \\ ((x + 1) + y)^* & = & \{y (x + 1) \leq (x + 1) y^* \text{ (below); separation (12)}\} \\ (x + 1)^* y^* & = & \{(x + 1)^* = x^*\} \\ x^* y^* & & \} \end{array}$$

$$\begin{array}{lcl} y (x + 1) & = & \{(\text{dist}) \} \\ y x + y & \leq & \{y x \leq (x + 1) y^* \text{ (hyp); } y \leq (x + 1) y^* \} \\ (x + 1) y^* & & \} \end{array}$$

□

The hypothesis of (16) cannot be weakened to  $y x \leq x^* y^*$ . For example, if  $y = \{\langle 0, 1 \rangle, \langle 1, 2 \rangle\}$  and  $x = \{\langle 1, 2 \rangle, \langle 2, 3 \rangle\}$ , then  $y x = \{\langle 0, 2 \rangle, \langle 1, 3 \rangle\} \leq x^* y^*$ , and  $\langle 0, 3 \rangle \in (x + y)^*$  but  $\langle 0, 3 \rangle \notin x^* y^*$ . Note also that the hypothesis of (16) do not work for  $\star$  iteration; for example, in a two-state relational model, if  $x = \{\langle 0, 1 \rangle\}$  and  $y = \{\langle 1, 0 \rangle\}$ , then  $(x + y) \star 0 = \{\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle\}$  but  $x \star y \star 0 = \{\}$ .

Combining (14) and the dual of (16) gives another theorem that works for both iterators:

$$(17) \quad y x \leq x x^* (1 + y) \implies (x + y) \circ z = x \circ y \circ z$$

$$\begin{array}{lcl} \text{true} & \implies & \{y x \leq x x^* (1 + y) \text{ (hyp)} \} \\ y x \leq x x^* (1 + y) & \implies & \{x x^* \leq x^*\} \\ y x \leq x^* (1 + y) & \implies & \{(16)' \text{ with } x, y := y, x \} \\ (x + y)^* z = x^* y^* z & \implies & \{y x \leq x (x + y)^* \text{ (hyp); (14)}\} \\ (x + y) \circ z = x \circ y \circ z & & \} \end{array}$$

□

Finally, more complex separation theorems are possible. For example:

$$(18) \quad y x \leq (x + z) y^* \wedge z x = 0 \implies (x + y + z) \circ u = x \circ (y + z) \circ u$$

$$\begin{array}{lcl} (x + y + z) \circ u & \leq & \{z \leq y^* z \} \\ (x + y^* z + y) \circ u & = & \{y (x + y^* z) \leq (x + y^* z) y^* \text{ (below); (12)}\} \\ (x + y^* z) \circ y \circ u & = & \{z x = 0 \text{ (hyp); so } (y^* z) x = 0; \text{ (12)} \} \\ x \circ (y^* z) \circ y \circ u & = & \{(y^* z) \circ y \circ u = (y + z) \circ u \text{ (4)} \} \\ x \circ (y + z) \circ u & & \} \end{array}$$

$$\begin{array}{lcl} y (x + y^* z) & = & \{(\text{dist}) \} \\ y x + y y^* z & \leq & \{y x \leq (x + z) y^* \text{ (hyp)} \} \\ (x + z) y^* + y y^* z & \leq & \{z \leq y^* z; y y^* z \leq y^* z y^*; \text{ (dist)} \} \\ (x + y^* z) y^* & & \} \end{array}$$

□

## 5 Reduction

The separation theorems of the last section can be viewed as special cases of theorems of the form

$$\dots \implies (x + y) \circ z = (p x + q y) \circ (\bar{p} x + \bar{q} y) \circ z$$

where  $p = 1$  and  $q = 0$ . In this section, we prove several such theorems, where the separands are obtained by prefixing or postfixing summands of the original iterand with predicates.

One way to interpret a theorem of the form above is as follows; other interpretations are given later in the section. Let  $x$  and  $y$  be programs with critical sections, let  $p$  mean that  $y$  is in a critical section and let  $q$  mean that  $x$  is in a critical section. The conclusion says that every concurrent execution of  $x$  and  $y$  is equivalent to one that executes in two phases: in the first phase,  $x$  executes only when  $y$  is in a critical section (and vice versa), and in the second phase,  $x$  executes only when  $y$  is not in a critical section (and vice versa). If the execution starts in a  $\bar{p} \bar{q}$  state (i.e., one where neither program is in its critical section), the first phase vanishes (thanks to loop elimination) and we have a corollary of the form

$$\dots \implies \bar{p} \bar{q} (x + y) \circ z = \bar{p} \bar{q} (\bar{p} x + \bar{q} y) \circ z$$

which says that we can pretend that critical sections are not interrupted. This amounts to pretending that critical sections are executed atomically.

### 5.1 Fair Progress Reduction

Consider a finite class of workers  $x_i$ , each of whom, at any time, are classified as “fast” ( $p_i$ ) or “slow” ( $\bar{p}_i$ ). The classification system can depend on the particular worker, so one worker might be classified as slow if anybody is waiting for him, while another might be classified as slow unless he’s waiting for somebody else. The only restriction we place on the classification is that if it is possible to execute a fast worker  $x_i$  followed by a slow worker  $x_j$ , then (1)  $x_j$  must already be classified as slow, and (2) the same effect can be obtained by executing  $x_j$  and then  $x_i$ .

The following theorem says that, under these conditions, any execution of the system is equivalent to one that executes in two phases, where only slow workers execute in the first phase and only fast workers execute in the second phase:

$$(19) \quad (\forall i : p_i x_i \bar{p}_j x_j \leq \bar{p}_j x_j x_i) \implies (+i : x_i) \circ y = (+i : \bar{p}_i x_i) \circ (+i : p_i x_i) \circ y$$

Let  $u = (+i : \overline{p}_i x_i)$ , and let  $v = (+i : p_i x_i)$ ; then  $(+i : x_i) = u + v$ , and

$$\begin{array}{rcl}
v u & = & \{(\text{dist})\} \\
(+i : p_i x_i \ (+j : \overline{p}_j x_j)) & = & \{(\text{dist})\} \\
(+i, j : p_i x_i \ \overline{p}_j x_j) & \leq & \{(\text{hyp})\} \\
(+i, j : \overline{p}_j x_j \ x_i) & = & \{(\text{dist})\} \\
(+j : \overline{p}_j x_j) \ (+i : x_i) & = & \{\text{def } u, v\} \\
u \ (u + v) & \leq & \{ \} \\
u \ u^* \ (1 + v) & & 
\end{array}$$

so  $(u + v) \circ y = u \circ v \circ y$  by (17)

□

For example, let the workers be connected by unbounded FIFO queues (at least one between every pair of distinct workers); at each step, a worker can receive any number of messages (but at most one from any queue) and send any number of messages. (As in all of our examples using queues, we assume that the only operations on queues are blocking send and receive operations.) Let  $p_i$  mean that every outgoing queue of  $x_i$  is longer than every incoming queue of  $x_i$ . The hypotheses of the theorem are then satisfied, because (1) at most one worker is fast at any time, and (2) if a worker is fast, all of his outgoing queues are nonempty, so  $p_i x_i \ \overline{p}_j x_j \leq x_j x_i$  (since the data manipulated by the  $x_j$  action is not modified by the  $x_i$  action).

An important special case is where there are exactly two workers, with a single FIFO queue (initially empty) in each direction, and at each step a worker either sends a single message or receives a single message. Then, in the first phase, the two queue lengths never differ by more than one, and so the phase can be parsed as a sequence of “macro steps”, each with one step for each worker. For many protocols, only a finite number of states are reachable in the first phase; since only one worker gets to execute in the second phase, a number of interesting properties (maximum queue length, freedom from deadlock, etc.) are decidable for such protocols; see [4].

## 5.2 One-way Reduction

A special case of fair progress reduction occurs when one of the workers ( $y$  below) is always slow:

$$(20) \quad p x \ \overline{p} = 0 \ \wedge \ p x \ y \leq y x \implies (x + y) \circ z = (\overline{p} x + y) \circ (p x) \circ z$$

(19), where  $i$  ranges over  $\{0, 1\}$ ,  $x_0 := x$ ,  $x_1 := y$ ,  $p_0 := p$ ,  $p_1 := 0$ .

□

For example, let  $y$  be (the transition relation of) a program with critical sections, where  $p$  means that control is inside a critical section. The hypotheses say that  $x$  cannot move control out of the critical section, and that, from inside the critical section,  $x$  actions commute to the right past  $y$  actions. The conclusion says that any execution is equivalent to one that executes in two phases: during

the first phase,  $x$  does not execute when control is in a critical section of  $y$  (i.e., critical sections execute atomically), and during the second phase,  $x$  executes exclusively, with control remaining in the critical section.

One-way reduction can be applied in many pipelining situations. For example, let  $x$  and  $y$  be programs that communicate via FIFO queues, and let  $p$  hold when every queue from  $x$  to  $y$  is nonempty. The hypotheses of (20) say that (1)  $x$  cannot empty a queue from  $x$  to  $y$  (i.e., messages cannot be unsent), and (2) if every queue from  $x$  to  $y$  is nonempty, then the result of performing an  $x$  step followed by a  $y$  step can be obtained by performing a  $y$  step first<sup>3</sup>. The conclusion says that we can pretend that execution proceeds in two phases; in the first phase,  $x$  executes only when one of his queues to  $y$  is empty, and in the second phase, only  $x$  gets to execute. In the special case where there is only one queue from  $x$  to  $y$ , the first phase condition says, in effect, that messages from  $x$  are received as soon as they are sent.

One shortcoming of fair progress reduction is that the  $\star$  part of the theorem can't be dualized. On the other hand, we can prove the dual of (20) as follows:

$$(21) \quad \bar{p} x p = 0 \wedge y x p \leq x y \implies (x + y) \circ z = (x p) \circ (y + x \bar{p}) \circ z$$

(18) with  $x := x p$ ,  $y := y + x \bar{p}$ ,  $z := x \bar{p}$ ,  $u := z$ , justified as follows:

$$\begin{array}{rcl} (y + x \bar{p}) (x p) & = & \{(\text{dist}) \quad \quad \quad \} \\ y x p + x \bar{p} x p & = & \{\bar{p} x p = 0 \text{ (hyp)} \quad \quad \} \\ y x p & \leq & \{y x p \leq x y \text{ (hyp)} \quad \quad \} \\ x y & \leq & \{x = x p + x \bar{p}; y \leq y + x \bar{p}\} \\ (x p + x \bar{p}) (y + x \bar{p}) & & \end{array}$$

$$\begin{array}{rcl} (x \bar{p}) (x p) & = & \{\bar{p} x p = 0 \text{ (hyp)} \quad \quad \} \\ 0 & & \end{array}$$

□

In contexts where control starts and ends outside of the critical section, we can eliminate the extra  $x$  iterations with loop elimination:

$$(22) \quad \bar{p} x p = 0 \wedge y x p \leq x y \implies \bar{p} (x + y) \circ z = \bar{p} (y + x \bar{p}) \circ z$$

$$(23) \quad p x \bar{p} = 0 \wedge p x y \leq y x \implies (x + y) \circ \bar{p} = (\bar{p} x + y) \circ (\bar{p} + (p x) \circ 0)$$

### 5.3 Two-phased reduction

Two-phased reductions combine (20) and (21). For this section and the next, we assume the following hypotheses:

$$\begin{array}{l} q y p = 0 \\ \bar{p} x p = 0 \\ q x \bar{q} = 0 \\ y x p \leq x y \\ \underline{q x y \leq y x} \end{array}$$

<sup>3</sup> This holds only if queues from  $y$  to  $x$  are unbounded, and  $y$  receives at most one message per queue per step.

These hypotheses can be read as follows. Think of the critical sections of  $y$  as operating in two phases, where  $p$  (resp.  $q$ ) means control is in the first (resp. second) phase. The first hypothesis says that control cannot pass directly from the second phase to the first phase without first leaving the critical section (this defines what we mean by a “two-phased” critical section). The next two hypotheses say that  $x$  cannot move control into the first phase or out of the second. The last two hypotheses say that  $x$  actions left-commute out of the first phase and right-commute out of the second phase. (Note that there are no constraints on an action that moves control from the first phase to the second phase.) The conclusion says that we can pretend that execution consists of a sequence of first phase  $x$  actions, followed by a phase where the critical section executes atomically, followed by a sequence of second phase  $x$  actions:

$$(24) \quad (x + y) \circ z = (x p) \circ (y + \bar{q} x \bar{p}) \circ (q x) \circ z$$

$$\begin{array}{lcl} (x + y) \circ z & \leq & \{(20) \} \\ (x p) \circ (y + x \bar{p}) \circ z & \leq & \{q (x \bar{p}) y \leq y (x \bar{p}) \text{ (below); (21)}\} \\ (x p) \circ (y + \bar{q} x \bar{p}) \circ (q x) \circ z & & \\ \\ q (x \bar{p}) y & \leq & \{q x \bar{q} = 0 \text{ (hyp)} \} \\ q x q y & \leq & \{q y p = 0 \text{ (hyp)} \} \\ q x y \bar{p} & \leq & \{q x y \leq y x \text{ (hyp)} \} \\ y x \bar{p} & & \} \end{array}$$

□

Again, by assuming control begins/ends outside of the critical section, we can eliminate the extra  $x$  iterations; for example,

$$(25) \quad \bar{p} (x + y) \circ \bar{q} = \bar{p} (\bar{q} x \bar{p} + y) \circ (\bar{q} + (q x) \circ 0)$$

Two-phased reduction is usually applied to pretend the atomicity of an operation that gathers information/resources, possibly executes some global atomic action, and then releases information/resources. Here are some examples of such operations (ignoring the subtleties of each case):

- a fragment of a sequential program that contains at most one access to a shared variable (the so-called single-assignment rule);
- a database transaction that that never obtains a lock after releasing a lock [9];
- a sequence of semaphore  $p$  operations followed by a sequence of semaphore  $v$  operations [8];
- a fragment of a distributed program that receives messages, performs a local action, and then sends messages [6].

#### 5.4 Lamport and Schneider’s Reduction Theorem

In traditional program reasoning, the concern is program properties rather than program refinement; to apply reduction in this context, we need a way to turn

properties of the “reduced” program (where critical sections execute atomically) into properties of the unreduced program. One way to do this is to restrict attention to states where control is outside the critical sections. For example, under the hypotheses of two-phased reduction, if  $Q$  always holds in the reduced program, then  $p + q + Q$  always holds in the unreduced program (i.e.,  $Q$  holds whenever control is outside the critical section). The early reduction theorems of Lipton [8] and Doeppner [3] had this structure. However, sometimes we want to show that  $Q$  holds in all states, even when control is inside the critical section.

A theorem of this kind was proposed by Lamport and Schneider [7]. In order to generalize their theorem to handle total correctness (with slightly stronger hypotheses), we write some of the hypotheses using  $\circ$  iteration, with the understanding that the same interpretation of  $\circ$  ( $*$  or  $\star$ ) is to be used for the hypotheses and theorem (26) below. Their hypotheses, suitably generalized, are as follows (in addition to the other hypotheses of section 5.3):

$$\begin{aligned}
I &= I \bar{p} \\
I (y + \bar{q} x \bar{p}) \circ \bar{p} \bar{q} \bar{Q} &= 0 \\
\bar{p} Q (y p)^* \bar{Q} &= 0 \\
\bar{Q} (q y)^* \bar{q} Q &= 0 \\
\bar{Q} 1^\omega &\leq \bar{Q} (q y)^* \bar{q} 1^\omega \\
(q x) \circ 0 &= 0
\end{aligned}$$

These hypotheses can be read as follows:

- the initialization code  $I$  leaves control outside of the first phase;
- $Q$  always holds in the reduced program whenever control is outside of the critical section; for  $\circ = \star$ , we also assume that the reduced program terminates;
- it is impossible to start outside the first phase and run first-phase actions so as to falsify  $Q$ ;
- it is impossible to run second-phase actions, ending outside the second phase, so as to truthify  $Q$ ; and
- from any state where control is in the second phase and  $Q$  is *false*, it is possible to execute second phase actions so as to bring control outside of the second phase<sup>4</sup>
- For  $\circ = \star$ ,  $x$  cannot execute forever with control in the second phase. (If  $\circ = *$ , the formula is trivially satisfied.)

The following theorem says that  $Q$  always holds in the unreduced program (and if  $\circ = \omega$ , the program terminates):

$$(26) \quad I (y + x)^* \bar{Q} = 0$$

<sup>4</sup> There are two ways to talk about possibility (i.e., relational totality) in omega algebras. The one used here is that  $x$  is total on a domain  $p$  iff  $p x 1^\omega = p 1^\omega$  (i.e. all effects of  $x$  can be later obliterated). This works fine for the relational model, and fits in well with proof checking mechanisms. An alternative, which works for other programming models, is that  $x$  is total on  $p$  iff, for every  $y$ ,  $y p x = 0 \implies y p = 0$ .

Define  $L = (y + \bar{q} x \bar{p})$ ; then

$$\begin{array}{ll}
I (y + x) \circ \bar{Q} & \leq \{1 \leq 1^\omega\} \\
I (y + x) \circ \bar{Q} 1^\omega & = \{\bar{Q} 1^\omega = \bar{Q} (q y)^* \bar{q} 1^\omega \text{ (hyp)}\} \\
I (y + x) \circ \bar{Q} (q y)^* \bar{q} 1^\omega & \leq \{\bar{Q} (q y)^* \bar{q} Q = 0 \text{ (hyp)}; \bar{Q} \leq 1\} \\
I (y + x) \circ (q y)^* \bar{q} \bar{Q} 1^\omega & \leq \{q y \leq y + x; u \circ u^* v = u \circ v\} \\
I (y + x) \circ \bar{q} \bar{Q} 1^\omega & = \{I = I \bar{p} \text{ (hyp)}; (25)\} \\
I \bar{p} L \circ (\bar{q} \bar{Q} 1^\omega + (q x) \circ 0) & = \{(q x) \circ 0 = 0 \text{ (hyp)}\} \\
I \bar{p} L \circ \bar{q} \bar{Q} 1^\omega & \leq \{\bar{p} u \circ v \leq u \circ \bar{p} (u p) \circ v \text{ (below)}\} \\
I L \circ \bar{p} (L p) \circ \bar{q} \bar{Q} 1^\omega & \leq \{L p = y p\} \\
I L \circ \bar{p} (y p) \circ \bar{q} \bar{Q} 1^\omega & \leq \{I L \circ (y p) \circ 0 \leq I L \circ 0 = 0 \text{ (hyp)}\} \\
I L \circ \bar{p} (y p)^* \bar{q} \bar{Q} 1^\omega & \leq \{\bar{p} Q (y p)^* \bar{Q} = 0 \text{ (hyp)}\} \\
I L \circ \bar{p} \bar{Q} (y p)^* \bar{q} \bar{Q} 1^\omega & \leq \{(y p) \bar{q} \leq y p \leq \bar{q} y \text{ (hyp)}; (\text{iter})'\} \\
I L \circ \bar{p} \bar{Q} \bar{q} y^* \bar{Q} 1^\omega & \leq \{I L \circ \bar{p} \bar{q} \bar{Q} = 0 \text{ (hyp)}\} \\
0 & \\
\bar{p} u \circ v & \leq \{u \leq u u^* \bar{p} + u p\} \\
\bar{p} (u u^* \bar{p} + u p) \circ v & \leq \{(u p) (u u^* \bar{p}) \leq (u u^* \bar{p}) 1; (12)\} \\
\bar{p} (u u^* \bar{p}) \circ (u p) \circ v & \leq \{\bar{p} (u u^* \bar{p}) \leq (u u^*) \bar{p}; (\text{iter})\} \\
(u u^*) \circ \bar{p} (u p) \circ v & = \{(u u^*) \circ w = u \circ w\} \\
u \circ \bar{p} (u p) \circ v & \\
\end{array}$$

□

## 5.5 Back's Theorem

Back[1, 2] proposed a reduction theorem for total correctness. Instead of breaking critical sections of  $y$  into phases, his theorem partitions the environment action  $x$  into two components,  $r$  and  $l$ ; inside a critical section,  $r$  (resp.  $l$ ) actions can always be pushed to the right (resp. left). Suitably generalized, his theorem uses the following hypotheses:

$$\begin{array}{l}
r p y \leq y r \\
y p l \leq l y \\
r p l \leq l r \\
p r \bar{p} = 0 \\
\bar{p} l p = 0
\end{array}$$

The first three conditions say that, inside of the critical section,  $r$  actions commute to the right of  $l$  and  $y$  actions, and  $l$  actions commute to the left of  $r$  and  $y$  actions. The last two conditions say that  $r$  (resp.  $l$ ) cannot move control out of (resp. in to) the critical section. The theorem says that we can pretend (in the main phase) that critical sections execute atomically:

$$(27) \quad (y + r + l) \circ z = (p l) \circ (y + \bar{p} l + r \bar{p}) \circ (r p) \circ z$$

$$\begin{aligned}
(y + r + l) \circ z &= \{l = p l + \bar{p} l \quad \} \\
(p l + y + r + \bar{p} l) \circ z &= \{(18) \text{ with } x := p l, y := y + r, \} \\
&\quad \{z := \bar{p} l, \text{ proofs (a),(b) below} \} \\
(p l) \circ (y + r + \bar{p} l) \circ z &= \{r = r p + r \bar{p} \quad \} \\
(p l) \circ (y + r \bar{p} + \bar{p} l + r p) \circ z &= \{\text{proof (c) below, (17)} \quad \} \\
(p l) \circ (y + r \bar{p} + \bar{p} l) \circ (r p) \circ z &
\end{aligned}$$

$$\begin{aligned}
\text{(a):} \\
(y + r + \bar{p} l) (p l) &= \{(\text{dist}) \quad \} \\
y p l + r p l + \bar{p} l p l &= \{\bar{p} l p = 0 \text{ (hyp)} \quad \} \\
y p l + r p l &= \{y p l \leq l y, r p l \leq l r \text{ (hyp)} \quad \} \\
l y + l r &\leq \{l = p l + \bar{p} l; (\text{dist}) \quad \} \\
(p l + \bar{p} l) (y + r) &
\end{aligned}$$

$$\begin{aligned}
\text{(b):} \\
(\bar{p} l) (p l) &= \{\bar{p} l p = 0 \text{ (hyp)} \quad \} \\
0 &
\end{aligned}$$

$$\begin{aligned}
\text{(c):} \\
(r p) (y + r \bar{p} + \bar{p} l) &= \{(\text{dist}) \quad \} \\
r p y + r p r \bar{p} + r p \bar{p} l &= \{p r \bar{p} = 0 \text{ (hyp); } p \bar{p} = 0 \quad \} \\
r p y &\leq \{r p y \leq y r \text{ (hyp)} \quad \} \\
y r &\leq \{r = r p + r \bar{p} \quad \} \\
(y + r \bar{p} + \bar{p} l) ((r p) + (y + r \bar{p} + \bar{p} l)) &
\end{aligned}$$

□

Again, assuming that  $r p$  actions terminate (i.e.,  $r$  actions cannot leave control inside the critical section forever), we obtain a simpler reduction theorem by loop elimination:

$$(28) \quad (r p) \circ 0 = 0 \implies \bar{p} (y + r + l) \circ \bar{p} = \bar{p} (y + \bar{p} l + r \bar{p}) \circ \bar{p}$$

A typical application of Back's theorem is the following asynchronous snapshot algorithm. Let  $y$  be a program whose critical section actions record values of some subset of the program variables, and let  $p$  mean that not all of these variables are recorded. (The critical section is entered by discarding all recorded values). Let  $r$  (resp.  $l$ ) be a sum of transitions that read and write variables in an arbitrary way, where each of these transitions can execute only if all (resp. none) of the variables it accesses are recorded. The hypotheses of Back's theorem then hold ( $r$  and  $l$  do not change the set of recorded variables, and so don't move control in or out of the critical section; each summand of  $r$  commutes with each summand of  $l$  because they must access disjoint sets of variables, etc.). Thus, we can pretend that the entire state is recorded without interruption.

## 6 Conclusion

In summary, separation theorems are powerful tools for reasoning about concurrency control. Our experience with reduction resulted in theorems that were

uniformly simpler, more powerful, and easier to prove than the originals. These theorems also show the advantages of working with simple iteration constructs (as opposed to the usual **do** or **while** loops), and of retaining sequential composition as a first-class operator.

## 7 Acknowledgments

Members of IFIP WG2.3, particularly J. R. Rao and Jay Misra, provided useful feedback on earlier versions of this work. The anonymous referees also provided a number of helpful suggestions.

## References

1. R. J. R. Back. Refining atomicity in parallel algorithms. In *Parallel Architectures and Languards Europe*. Springer-Verlag, 1989.
2. R. J. R. Back and J. von Wright. Reasoning algebraically about loops. *Acta Informatica*, 36(4), 1999.
3. T. Doeppner. Parallel program correctness through refinement. In *ACM Symposium on Principles of Programming Languages*, 1977.
4. M. G. Gouda and Y. T. Yu. Protocol validation by maximal progress state exploration. Technical Report CS-TR-211, The University of Texas at Austin, 1982.
5. D. Kozen. A completeness theorem for Kleene algebra and the calculus of regular events. *Information and Computation*, 110(2):366–390, 1994.
6. L. Lamport. A theorem on atomicity in distributed algorithms. *Distributed Computing*, 4:59–68, 1990.
7. L. Lamport and F. B. Schneider. Pretending atomicity. Technical Report Research Report 44, Compaq System Research Center, 1989.
8. R. J. Lipton. Reduction: A method of proving properties of parallel programs. *Communications of the ACM*, 18(12):717–721, 1975.
9. C. Papadimitriou. *The Theory of Database Concurrency Control*. Computer Science Press, 1986.