Texts and Monographs in Computer Science

Suad Alagic, **Object-Oriented Database Programming**

Suad Alagic, **Relational Database Technology**

Suad Alagic and Michael A. Arbib, **The Design of Well-Structured and Correct Programs**

S. Thomas Alexander, **Adaptive Signal Processing: Theory and Applications**

Krzysztof R. Apt and Ernst-Rüdiger Olderog, **Verification of Sequential and Concurrent Programs**

Michael A. Arbib, A.J. Kfoury, and Robert N. Moll, **A Basis for Theoretical Computer Science**

Friedrich L. Bauer and Hans Wössner, **Algorithmic Language and Program Development**

W. Bischofberger and G. Pomberger, **Prototyping-Oriented Software Development: Concepts and Tools**

Ronald V. Book and Friedrich Otto, **String-Rewriting Systems**

Kaare Christian, **A Guide to Modula-2**

Edsger W. Dijkstra, **Selected Writings on Computing: A Personal Perspective**

Edsger W. Dijkstra and Carel S. Scholten, **Predicate Calculus and Program Semantics**

W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and J. Misra, Eds., **Beauty Is Our Business: A Birthday Salute to Edsger W. Dijkstra**

P.A. Fejer and D.A. Simovici, **Mathematical Foundations of Computer Science, Volume I: Sets, Relations, and Induction**

Melvin Fitting, **First-Order Logic and Automated Theorem Proving**

Nissim Francez, **Fairness**

R.T. Gregory and E.V. Krishnamurthy, **Methods and Applications of Error-Free Computation**

David Gries, Ed., **Programming Methodology: A Collection of Articles by Members of IFIP WG2.3**

David Gries, **The Science of Programming**

David Gries and Fred B. Schneider, **A Logical Approach to Discrete Math**

(continued after index)

The Science of Programming

David Gries

(on "guarded if")

© 1981
Springer-Verlag
New York Berlin Heidelberg London Paris
Tokyo Hong Kong Barcelona Budapest

Exercises for Section 9.4

1. Transform the following using definition (9.4.7) so that they denote conventional textual substitution —i.e. the superscript in each is a list of distinct identifiers.

(a) $R_{e,f,g}^{b[i],b[j],x}$

(b) $R_{e,f,g}^{b[i],x,b[j]}$

(c) $R_{e,f,g}^{b[i],c[i],b[j]}$

(d) $R_{e,f,g,h}^{b[i],c[i],b[j],c[j]}$

2. Determine and simplify the following weakest preconditions, where b is an array of integers and it is assumed that all subscripts are in range.

(a) $wp(\text{“}b[i],b[2]:=3,4\text{”},\ b[i]=3)$

(b) $wp(\text{“}b[i],b[2]:=4,4\text{”},\ b[i]=3)$

(c) $wp(\text{“}p,\ b[p]:=b[p],p\text{”},\ p=b[p])$

(d) $wp(\text{“}i,\ b[i]:=i+1,0\text{”},\ 0<i \wedge (A\,j:0\leqslant j<i:b[j]=0))$

(e) $wp(\text{“}i,\ b[i]:=i+1,0\text{”},\ 0<i \wedge b[0:i-1]=0)$

(f) $wp(\text{“}p,\ b[p],b[q]:=b[p],b[q],p\text{”},\ p=b[q])$

(g) $wp(\text{“}p,\ b[p],b[b[p]]:=b[p],b[b[p]],p\text{”},\ p=b[b[p]])$

(h) $wp(\text{“}p,\ b[p],b[b[p]]:=b[p],b[b[p]],p\text{”},\ p\neq b[b[p]])$

3. Prove the following implication:

$$i=I \wedge b[i]=K \Rightarrow wp(\text{“}i,\ b[i]:=b[i],i\text{”},\ i=K \wedge b[I]=I)$$

4. Derive a definition for a general multiple assignment command that can include assignments to simple variables, array elements and Pascal record fields. (see exercise 1 of section 5.3.)

5. Prove that lemma 4.6.3 holds for the extended definition of textual substitution:

Lemma. Suppose each x_i of list $\bar{x}$ has the form *identifier* ∘ *selector* and suppose $\bar{u}$ is a list of fresh, distinct identifiers. Then

$$(E_{\bar{u}}^{\bar{x}})_{\bar{x}}^{\bar{u}} = E \quad \square$$

Chapter 10
The Alternative Command

Programming notations usually have a conditional command, or **if**-statement, which allows execution of a subcommand to be dependent on the current state of the program variables. An example of a conditional command, taken from ALGOL 60 and Pascal, is

if $x \geqslant 0$ **then** $z := x$ **else** $z := -x$

Execution of this command stores the absolute value of x in z: if $x \geqslant 0$ then the first alternative $z := x$ is executed; otherwise the second alternative $z := -x$ is executed. In our programming notation, this command can be written as

(10.1)
if $x \geqslant 0 \rightarrow z := x$
[] $x \leqslant 0 \rightarrow z := -x$
fi

or, since it is short and simple enough, on one line as

if $x \geqslant 0 \rightarrow z := x$ [] $x \leqslant 0 \rightarrow z := -x$ **fi**

Command (10.1) contains two entities of the form $B \rightarrow S$ (separated by the symbol []) where B is a Boolean expression and S a command. $B \rightarrow S$ is called a *guarded command*, for B acts as a guard at the gate $\rightarrow$, making sure S is executed only under the right conditions. To execute (10.1), find one true guard and execute its corresponding command. Thus, with $x > 0$ execute $z := x$, with $x < 0$ execute $z := -x$, and with $x = 0$ execute *either* (but not both) of the assignments.

This brief introduction has glossed over a number of important points. Let us now be more precise in describing the syntax and execution of the alternative command.

The general form of the alternative command is

(10.2) **if** $B_1 \to S_1$
$\quad [\!] B_2 \to S_2$
$\quad \cdots$
$\quad [\!] B_n \to S_n$
fi

where $n \geqslant 0$ and each $B_i \to S_i$ is a guarded command. Each S_i can be *any* command —*skip*, *abort*, sequential composition, assignment, another alternative command, etc.

For purposes of abbreviation, we refer to the general command (10.2) as IF, while BB denotes the disjunction

$$B_1 \vee B_2 \vee \cdots \vee B_n.$$

Command IF can be executed as follows. First, if any guard B_i is not well-defined in the state in which execution begins, abortion may occur. This is because nothing is assumed about the order of evaluation of the guards. Secondly, at least one guard must be true; otherwise execution aborts. Finally, if at least one guard is true, then one guarded command $B_i \to S_i$ with true guard B_i is chosen and S_i is executed. (skip to p. 134

The definition of $wp(\text{IF}, R)$ is now quite obvious. The first conjunct indicates that the guards must be well-defined. The second conjunct indicates that at least one guard is true. The rest of the conjuncts indicate that execution of each command S_i with a true guard B_i terminates with R true:

(10.3a) **Definition.** $wp(\text{IF}, R) = domain(\text{BB}) \wedge \text{BB} \wedge$
$\quad (B_1 \Rightarrow wp(S_1, R)) \wedge \cdots \wedge (B_n \Rightarrow wp(S_n, R))$ □

Typically, we assume that the guards are total functions —i.e. are well-defined in all states. This allows us to simplify the definition by deleting the first conjunct. Thus, with the aid of quantifiers we rewrite the definition in (10.3b) below. From now on, we will use (10.3b) as the definition, but be sure the guards are well-defined in the states in which the alternative command will be executed!

(10.3b) **Definition.** $wp(\text{IF}, R) = (E\, i: 1 \leqslant i \leqslant n: B_i) \wedge$
$\quad (A\, i: 1 \leqslant i \leqslant n: B_i \Rightarrow wp(S_i, R))$ □

Example 1. Let us show that, under all initial conditions, execution of (10.1) stores the absolute value of x in z. That is, we want to show that $wp((10.1), z = abs(x)) = T$. We have:

$$\begin{aligned}
&wp((10.1), z = abs(x)) \\
&\quad = (x \geqslant 0 \vee x \leqslant 0) \wedge \\
&\qquad (x \geqslant 0 \Rightarrow wp(\text{“}z := x\text{”}, z = abs(x))) \wedge \\
&\qquad (x \leqslant 0 \Rightarrow wp(\text{“}z := -x\text{”}, z = abs(x))) \\
&\quad = T \wedge (x \geqslant 0 \Rightarrow x = abs(x)) \wedge \\
&\qquad (x \leqslant 0 \Rightarrow -x = abs(x)) \\
&\quad = T \wedge T \wedge T \\
&\quad = T \quad \square
\end{aligned}$$

$$\left\{\begin{array}{l} BB \wedge \\ B_1 \Rightarrow wp(S_1, R) \wedge \\ B_2 \Rightarrow wp(S_2, R) \end{array}\right.$$

Example 2. The following command is supposed to be the body of a loop that counts the number of positive values (p) in array $b[0{:}m-1]$.

(10.4) **if** $b[i] > 0 \to p, i := p+1, i+1$
$\quad [\!] b[i] < 0 \to i := i+1$
fi

After execution of this command we expect to have $i \leqslant m$ and p equal to the number of values in $b[0{:}i-1]$ that are greater than zero. Letting R be the assertion

$$i \leqslant m \wedge p = (N\, j: 0 \leqslant j < i: b[j] > 0)$$

we calculate:

$$\begin{aligned}
&wp((10.4), R) = (b[i] > 0 \vee b[i] < 0) \wedge \\
&\qquad\qquad (b[i] > 0 \Rightarrow wp(\text{“}p, i := p+1, i+1\text{”}, R)) \wedge \\
&\qquad\qquad (b[i] < 0 \Rightarrow wp(\text{“}i := i+1\text{”}, R) \\
&= b[i] \neq 0 \wedge \\
&\quad (b[i] > 0 \Rightarrow i+1 \leqslant m \wedge p+1 = (N\, j: 0 \leqslant j < i+1: b[j] > 0)) \wedge \\
&\quad (b[i] < 0 \Rightarrow i+1 \leqslant m \wedge p = (N\, j: 0 \leqslant j < i+1: b[j] > 0)) \\
&= b[i] \neq 0 \wedge i < m \wedge \\
&\quad p = (N\, j: 0 \leqslant j < i: b[j] > 0) \wedge \\
&\quad p = (N\, j: 0 \leqslant j < i: b[j] > 0) \\
&= b[i] \neq 0 \wedge i < m \wedge p = (N\, j: 0 \leqslant j < i: b[j] > 0)
\end{aligned}$$

Hence we see that array b should not contain the value 0, and that the definition of p as the number of values greater than zero in $b[0{:}i-1]$ will be true after execution of the alternative command if it is true before. □

The reader may feel that there was too much work in proving what we did in example 2. After all, the result can be obtained in an intuitive manner, and perhaps fairly easily (although one is likely to overlook the problem with zero elements in array b). At this point, it is important to practice such formal manipulations. It results in better understanding of the theory and better understanding of the alternative command itself.

Moreover, the kind of manipulations performed in example 2 will indeed be necessary in developing some programs, and the facility needed for this can only come through practice. Even the act of performing a few exercises will begin to change the way you "naturally" think about programs and thus what you call your intuition about programming.

Later on, when attacking a problem that is similar to one worked on earlier, it may not be necessary to be so formal, but the formality will be at your fingertips when you need it on the more difficult problems.

Some comments about the alternative command

The alternative command differs from the conventional **if**-statement in several respects. We now discuss the reasons for these differences.

First, the alternative command allows any number of alternatives, not just two. Thus, it serves also as a "case statement" (Pascal) or "SELECT statement" (PL/I). There is no need to have two different notations, one for two alternatives and one for more. One notation for one concept —in this case alternation or choice— is a well-known, reasonable principle.

There are no defaults: each alternative command must be preceded by a guard that describes the conditions under which it may be executed. For example, the command to set x to the absolute value of x must be written with two guarded commands:

$$\textbf{if } x \geqslant 0 \rightarrow \mathit{skip} \ [] \ x \leqslant 0 \rightarrow x := -x \ \textbf{fi}$$

Its counterpart in ALGOL, **if** $x<0$ **then** $z := -x$, has the default that if $x \geqslant 0$ execution is equivalent to execution of *skip*. Although a program may be a bit longer because of the lack of a default, there are advantages. The explicit appearance of each guard does aid the reader; each alternative is given in full detail, leaving less chance of overlooking something. More importantly, the lack of a default helps during program development. Upon deriving a possible alternative command, the programmer is *forced* to derive the conditions under which its execution will perform satisfactorily and, moreover, is *forced* to continue deriving alternatives until at least one is true in each possible initial state. This point will become clearer in Part III.

The absence of defaults introduces, in a reasonable manner, the possibility of nondeterminism. Suppose $x=0$ when execution of command (10.1) begins. Then, since both guards $x \geqslant 0$ and $x \leqslant 0$ are true, *either* command may be executed (but only one of them). The choice is entirely up to the executor —for example it could be a random choice, or on days with odd dates it could be the first and on days with even dates it could be the second, or it could be chosen to minimize execution time. The point is that, since execution of either one leads to a correct result, the programmer should not have to worry about which one is executed. He is free to derive as many alternatives commands and corresponding guards as possible, without regard to overlap.

Of course, for purposes of efficiency the programmer could strengthen the guards to excise the nondeterminism. For example, changing the second guard in (10.1) from $x \leqslant 0$ to $x<0$ would help if evaluation of unary minus is expensive, because in the case $x=0$ only the first command $z := x$ could then be executed.

Finally, the lack of default allows the possibility of symmetry (see (10.1)), which is pleasing —if not necessary— to one with a mathematical eye.

A theorem about the alternative command

Quite often, we are not interested in the weakest precondition of an alternative command, but only in determining if a known precondition implies it. For example, if the alternative command appears in a program, we may already know that its precondition is the postcondition of the previous command, and we really don't need to calculate the weakest precondition. In such cases, the following theorem is useful.

(10.5) **Theorem.** Consider command IF. Suppose a predicate Q satisfies

(1) $Q \Rightarrow \mathrm{BB}$
(2) $Q \wedge B_i \Rightarrow wp(S_i, R)$, for all i, $1 \leqslant i \leqslant n$.

Then (and only then) $Q \Rightarrow wp(\mathrm{IF}, R)$. □

Proof. We first show how to take Q outside the scope of the quantification in assumption 2 of the theorem:

$$\begin{aligned}
&(\mathbf{A}\, i: Q \wedge B_i \Rightarrow wp(S_i, R)) \\
&\quad = (\mathbf{A}\, i: \neg(Q \wedge B_i) \vee wp(S_i, R)) && \text{(Implication)} \\
&\quad = (\mathbf{A}\, i: \neg Q \vee \neg B_i \vee wp(S_i, R)) && \text{(De Morgan)} \\
&\quad = \neg Q \vee (\mathbf{A}\, i: \neg B_i \vee wp(S_i, R)) && (Q \text{ doesn't depend on } i) \\
&\quad = Q \Rightarrow (\mathbf{A}\, i: B_i \Rightarrow wp(S_i, R)) && \text{(Implication, twice)}
\end{aligned}$$

Hence, we have

$$\begin{aligned}
&(Q \Rightarrow \mathrm{BB}) \wedge (\mathbf{A}\, i: Q \wedge B_i \Rightarrow wp(S_i, R)) && \text{(Assumptions (1), (2))} \\
&\quad = (Q \Rightarrow \mathrm{BB}) \wedge (Q \Rightarrow (\mathbf{A}\, i: B_i \Rightarrow wp(S_i, R)) && \text{(From above)} \\
&\quad = Q \Rightarrow (\mathrm{BB} \wedge (\mathbf{A}\, i: B_i \Rightarrow wp(S_i, R)) \\
&\quad = Q \Rightarrow wp(\mathrm{IF}, R) && \text{(Definition (10.3b))}
\end{aligned}$$