On the Completeness of Test Cases for Atomic Arithmetic Expressions *

T. H. Tse[†]
The University of Hong Kong

T. Y. Chen
Swinburne University of Technology, Australia

X. Feng
The University of Hong Kong

Abstract

Most research on weak mutation testing focuses on predicate statements. Relative little attention has been paid to arithmetic expressions. In this paper, we analyse the latter type of expression and prove that, given an atomic arithmetic expression, if it contains no variable or if the operator is the unary " ++ " or " −− ", then a single test case is sufficient and necessary to kill any fundamental mutant; otherwise, two test cases are sufficient and necessary.

Keywords: Arithmetic expressions, completeness of test cases, mutation operators, mutation testing, software testing.

1. Introduction

Since mutation testing was proposed as an effective method for software testing, there have been many improvements. Howden [4] classified mutation testing techniques into two categories: strong mutation testing and weak mutation testing. Strong mutation testing creates mutants for a complete program and test cases are generated to differentiate the program from its mutants. Weak mutation testing only creates mutants for a statement and test cases are generated to differentiate the statement from its mutants.

In this paper, we are interested in weak mutation testing. We have found that most of the work in this area, such as [4] and [7], are concerned about predicate statements only. Relatively little work has been done on arithmetic expressions. The work of Foster [2], for instance, is rather limited in scope.

There are two opposing thoughts on the testing of arithmetic expressions: it is considered either very simple or very complex. On one hand, some testers think that only one test case is sufficient. Typical examples are the code coverage methods [3] such as path testing, branch testing and statement testing. On the other hand, other testers consider the testing of arithmetic expression testing to be very complex because of an infinitely large input space and unfathomable possibilities of errors. Complex recommendations involving, for example, boundary values [5, 9] are often given.

In this paper, we will discuss the test case generation for atomic arithmetic expression with no more than one operator. We shall formally prove that, given an atomic arithmetic expression, if it contains no variable or if the operator is the unary " ++ " or " −− ", then a single test case is sufficient and necessary to kill any fundamental mutant; otherwise, two test cases are sufficient and necessary.

Our paper is grouped as follows: Section 2 gives some background of our analysis. Sections 3 and 4 discuss test cases for VDTR, Uuor and ORAN mutants, respectively. Section 5 gives the concluding remarks.

2. Background

In weak mutation testing, a mutant is produced by replacing operands or operators in a statement. There are an infinite number of possible replacements. Do we need to test all possible mutants for each expression? Offutt [6] pointed out that it would be redundant to test every mutant. In most cases, test cases that can detect one mutant are also effective in detecting others. Only five mutation operators are found to cover more than 99 percent of the situations in Fortran 77. Agrawal [1] figured out similar mutation operators for C programs. By comparing C, Fortran 77 and Ada, Voas [8] also highlighted the same five mutation operators. Thus, in our research on the completeness of test cases, we shall only concentrate on the following five fundamental mutation operators:

*This research is supported in part by the Hong Kong Research Grants Council and the University of Hong Kong Committee on Research and Conference Grants.

[†]All correspondence should be addressed to Dr. T. H. Tse, Department of Computer Science and Information Systems, The University of Hong Kong, Pokfulam Road, Hong Kong. Email: "tse@csis.hku.hk".

Uuor: Insert unary operators in front of expressions.

VDTR: Force each arithmetic expression to take on a positive value, a negative value and zero.

ORAN: Replace each arithmetic operator with every syntactically legal operator.

ORRN: Replace relational operators with other relational operators.

OBBN: Replace logical connector with every syntactically logical operator.

Among the five, only Uuor, VDTR and ORAN mutation operators are related to arithmetic expressions. Hence, we shall concentrate on these three in the current paper.

An atomic expression is an expression with no more than one operator. The following are the operators: +, −, *, /, %, =, ++ and −−. According to the number of operators and operands, atomic arithmetic expressions can be divided into four categories: constants, simple variables, binary operator expressions and unary operator expressions. In this paper, we also assume that there is only one fault in an expression.

3. Complete Test Cases for VDTR and Uuor Mutants

The conditions we give for the generation of test cases for one mutant are all sufficient and necessary. Test cases that satisfy the conditions can be used to detect the mutant. On the other hand, test cases that do not satisfy these conditions cannot be used to detect the mutants.

Theorem 1
At most two test cases are needed to distinguish an atomic expression from its VDTR and Uuor mutants.

Proof
We shall discuss VDTR and Uuor mutants for constants, simple variables, binary operators and unary operators separately.

(*a*) Constants

If a constant is replaced by another, a single test case is sufficient. Any values can be taken as a test case.

(*b*) Simple Variable Expressions of the Form "*x*"

Its VDTR mutant is K, where K is a constant. Its Uuor mutant is $-x$. Obviously, two test cases $\{v_1, v_2\}$ such that $v_1 \neq v_2$ guarantee that the expression can be differentiated from its VDTR and Uuor mutants. On the other hand, a single test case "v" cannot distinguish the statement "$z = x;$" from the mutant "$z = v;$". Hence, a single test case will not be sufficient to detect all the mutants.

	Statement	**Results of Test Case $\mathbf{t_1 : (u_1, v_1)}$**	**Results of Test Case $\mathbf{t_2 : (u_2, v_2)}$**
Original	$\mathbf{x+y}$	$\mathbf{(u_1+v_1)}$	$\mathbf{(u_2+v_2)}$
Mutants	$x+K\ (K=v_1)$	(u_1+v_1)	(u_2+v_1)
	$x+K\ (K\neq v_1)$	(u_1+K)	(u_2+K)
	$K+y\ (K=u_1)$	(u_1+v_1)	(u_1+v_2)
	$K+y\ (K\neq u_1)$	$(K+v_1)$	$(K+v_2)$
	$-x+y$	$-u_1+v_1$	$-u_2+v_2$
	$x+-y$	u_1+-v_1	u_2+-v_2

Table 1. VDTR and Uuor mutants

(*c*) Binary Operator Expressions of the Form "$x \odot y$"

The VDTR mutants for the expression $x \odot y$ are $x \odot K$ and $K \odot y$, where K is a constant. The Uuor mutants are $-x \odot y$ and $x \odot -y$. It is obvious that the use of one test case will not be sufficient to detect all these mutants, since the test case $(u,\ v)$ cannot differentiate $x \odot y$ from the mutants $u \odot y$ and $x \odot v$.

The VDTR and Uuor mutants of the expression $x+y$ are shown in Table 1.

For the expression $x+y$, the test cases t_1 and t_2 can differentiate $x+y$ from its VDTR and Uuor mutants.

(*i*) $x+y, x-y, x*y$ and $x\,/\,y$ (float)

For expression $x+y$, we generate two test cases $t_1 : (u_1,\ v_1)$, $t_2 : (u_2,\ v_2)$. They satisfy the following conditions:

$$(C1):\quad u_1 \neq u_2 \text{ and } v_1 \neq v_2$$
$$(C2):\quad u_i \neq 0 \text{ and } v_i \neq 0 \text{ for } i = 1 \text{ or } 2$$

Test cases that satisfy $(C1)$ can detect VDTR mutants. Test cases that does not satisfy $(C1)$ cannot detect the mutant $u_1 + y$ or $x + v_1$.

For the condition $(C2)$, test cases such that $u_i \neq 0$ and $v_i \neq 0$ can detect Uuor mutants. A test case that does not satisfy this condition cannot detect a Uuor mutant.

The same results can be obtained for the operators " − ", " * " and " / " (float). However, for the operators " / " (int) and " % ", generating test cases for VDTR mutants is slightly more complex.

" % " and " / " are two special operators. Condition $(C1)$ is not sufficient to detect VDTR mutants for such expressions. The following is a brief discussion about the conditions of test cases to detect their VDTR mutants.

(ii) ***x % y***

If the operator is "%", two kinds of VDTR mutant are $K \% y$ and $x \% K$, where K is some integer. Suppose $t_1 : (u_1, v_1)$ and $t_2 : (u_2, v_2)$ are two test cases. Let $m_1 = u_1 \% v_1$, $n_1 = u_1 / v_1$, $m_2 = u_2 \% v_2$ and $n_2 = u_2 / v_2$. For the mutant $K \% y$,

$$\begin{aligned}&\text{if } u_1 \% v_1 = K \% v_1, \text{ then } K = m_1 + p_1 v_1\\ &\text{if } u_2 \% v_2 = K \% v_2, \text{ then } K = m_2 + p_2 v_2\end{aligned}$$

for some integers p_1 and p_2. A sufficient and necessary condition for t_1 and t_2 to kill this mutant is $m_1 + v_1 p_1 \neq m_2 + v_2 p_2$.

For the mutant $y \% K$, where K is a constant, let $n_1 = u_1 / v_1$, $n'_1 = u_1 / K$, $n_2 = u_2 / v_2$ and $n'_2 = u_2 / K$. We have:

$$\begin{aligned}&\text{if } u_1 \% v_1 = u_1 \% K, \text{ then } K = v_1 n_1 / n'_1\\ &\text{if } u_2 \% v_2 = u_2 \% K, \text{ then } K = v_2 n_2 / n'_2\end{aligned}$$

If two test cases cannot detect the mutant, then $v_1 n_1 / n'_1 = v_2 n_2 / n'_2$ and $|K| > \max(m_1, m_2)$. At least one test case can detect the mutant if they satisfy the conditions $v_1 n_1 / n'_1 = v_2 n_2 / n'_2 = K$ and $K \leq \max(m_1, m_2)$. In other words, $K = \mathrm{GCD}(v_1 n_1, v_2 n_2) = 1$ or $|K| \leq \max(|m_1|, |m_2|)$. That is, $\mathrm{GCD}(v_1 n_1, v_2 n_2) \leq \max(|m_1|, |m_2|)$.

Obviously, the deduction process is reversible. Therefore, the sufficient and necessary condition to distinguish $x \% y$ from its VDTR mutants is

$$\begin{aligned}&m_1 - m_2 \neq v_2 p_2 - v_1 p_1 \text{ and}\\ &\mathrm{GCD}(v_1 n_1, v_2 n_2) \leq \max(m_1, m_2)\end{aligned}$$

where m_1, m_2, n_1, n_2, p_1 and p_2 are as defined above.

(iii) ***x / y***

For the operator "/", two kinds of VDTR mutant are x / K and K / y, where K is a constant. Let $x_1 / y_1 = n_1$, $x_2 / y_2 = n_2$, $x_1 \% y_1 = m_1$ and $x_2 \% y_2 = m_2$.

For the mutant x / K,

$$\begin{aligned}&\text{if } x_1 / y_1 = x_1 / K, \text{ then}\\ &\quad x_1 = n_1 K + m_1 \text{ and } 0 \leq m_1 < K \qquad (1)\\ &\text{if } x_2 / y_2 = x_2 / K, \text{ then}\\ &\quad x_2 = n_2 K + m_2 \text{ and } 0 \leq m_2 < K \qquad (2)\end{aligned}$$

For (1), $x_1 / (n_1 + 1) < K \leq x_1 / n_1$. For (2), $x_2 / (n_2 + 1) < K \leq x_2 / n_2$. A test case that can detect the mutant must satisfy the condition $x_1 / n_1 \leq x_2 / (n_2 + 1)$.

For the mutant K / y,

$$\begin{aligned}&\text{if } x_1 / y_1 = K / y_1, \text{ then}\\ &\quad K = n_1 y_1 + m_1 \qquad (3)\\ &\text{if } x_2 / y_2 = K / y_2, \text{ then}\\ &\quad K = n_2 y_1 + m_2 \qquad (4)\end{aligned}$$

For (3), $n_1 y_1 \leq K < (n_1 + 1) y_1$. For (4), $n_2 y_2 \leq K < (n_2 + 1) y_2$. Hence, $n_2 y_2 \geq (n_1 + 1) y_1$ or $n_1 y_1 \geq (n_2 + 1) y_2$. The sufficient and necessary condition for detecting VDTR mutants of x / y is $(x_1 / n_1 \leq x_2 / (n_2 + 1))$ and $(n_2 y_2 \geq (n_1 + 1) y_1$ or $n_1 y_1 \geq (n_2 + 1) y_2)$.

(iv) ***x = y***

If the operator is "=", its VDTR mutant can only be $x = K$ and its Uuor mutant is $x = -y$. Obviously, two test cases are needed.

(d) **Unary Operator Expressions of the Form "$x \odot$" or "$\odot x$"**

For unary operator expressions of the Form "$x\odot$", the operator can only be "++" or "−−", in which case the expression has no VDTR or Uuor mutants. For unary operator expressions of the Form "$\odot x$", if the operator is "−", two test cases $\{v_1, v_2\}$ such that $v_1 \neq v_2$ are sufficient. Otherwise the expression has no VDTR or Uuor mutants. ■

We can see that, for atomic arithmetic expressions, two test cases are sufficient to differentiate an expression from its possible VDTR and Uuor mutants. In general, if an atomic arithmetic expression contains variables and the operator is not "++" or "−−", then two test cases are necessary. Otherwise, a single test case is sufficient.

4. Complete Test Cases for ORAN Mutants

There is no operator in constants or simple variables, and hence there is no ORAN mutant for such cases. In this section, therefore, we shall only discuss atomic arithmetic expressions with one operator.

Theorem 2
Given any atomic arithmetic expression, one test case can be found to distinguish it from its all ORAN mutants.

Proof
First, consider atomic arithmetic expressions with binary operators. All the possible binary arithmetic operators are listed in the seven rows of Table 2. The possible VDTR mutants are expressions with the original operators replied by other arithmetic operators, logical operators and relational operators. The former two kinds of replacement are shown in the nine columns of Table 2. The last kind of replacement will be discussed at the end of the proof.

	$x+y$	$x-y$	$x*y$	$x\ /\ y$ (float)	$x\ /\ y$ (int)	$x\ \%\ y$ (int)	$x=y$	$x\ \&\&\ y$	$x\ \|\|\ y$
$x+y$		$(A1)$	$(A2)$	$(A3)$	$(A4)$	$(A5)$	$(A6)$	$(A7)$	$(A8)$
$x-y$	$(B1)$		$(B2)$	$(B3)$	$(B4)$	$(B5)$	$(B6)$	$(B7)$	$(B8)$
$x*y$	$(C1)$	$(C2)$		$(C3)$	$(C4)$	$(C5)$	$(C6)$	$(C7)$	$(C8)$
$x\ /\ y$ (float)	$(D1)$	$(D2)$	$(D3)$		$(D4)$	$(D5)$	$(D6)$	$(D7)$	$(D8)$
$x\ /\ y$ (int)	$(E1)$	$(E2)$	$(E3)$	$(E4)$		$(E5)$	$(E6)$	$(E7)$	$(E8)$
$x\ \%\ y$ (int)	$(F1)$	$(F2)$	$(F3)$	$(F4)$	$(F5)$		$(F6)$	$(F7)$	$(F8)$
$x=y$	$(G1)$	$(G2)$	$(G3)$	$(G4)$	$(G5)$	$(G6)$		$(G7)$	$(G8)$

Table 2. ORAN mutants

(a) $\boldsymbol{x+y}$

Row 1 of Table 2 summarizes the sufficient and necessary conditions for detecting the listed ORAN mutants of the expression $x+y$. The details are as follows:

$(A1)$: $v \neq 0$
$(A2)$: $u \neq v\ /\ (v-1)$
$(A3)$: $u \neq v^2\ /\ (1-v)$
$(A4)$: $(u \neq -v-1$ or $v<0)$ and $(v \neq 2$ or $u \neq -4)$
$(A5)$: $u*v>0$, $|u| \geq 2|v|$ or $|u|<|v|$
$(A6)$: $u \neq 0$
$(A7)$: $(u+v \neq 1$ or $u=0$ or $v=0)$ and $(u \neq 0$ or $v \neq 0)$
$(A8)$: $u+v \neq 1$ and $(u \neq 0$ or $v \neq 0)$

$(A1)$, $(A2)$ and $(A3)$ are obviously the sufficient and necessary conditions for the mutants $x-y$, $x*y$ and $x\ /\ y$ (float), respectively.

$(A4)$ is used to distinguish the mutant $x\ /\ y$ (int) from $x+y$. We know that, if $x+y=x\ /\ y$ (int), then $x=(x+y)y+n$, giving $n=x-(x+y)y$. If $x \geq 0$, then $0 \leq n<|y|$. If $x<0$, then $-|y|<n \leq 0$. We have the following two cases:

(i) $\boldsymbol{x \geq 0}$

If $y>0$, then $x-(x+y)y \geq 0$, giving $x \geq (x+y)y$. Obviously, it is impossible.

If $y<0$, then $0 \leq x-(x+y)y<-y$, giving $-(y+1)+1\ /\ (1-y) \leq x \leq -(y+1)$, which is also impossible.

(ii) $\boldsymbol{x<0}$

If $y>0$, then $-y<x-(x+y)y \leq 0$, giving $-y-1+1\ /\ (1-y) \leq x<-y$, which means $(x=-y-2$ and $y=2)$ or $x=-y-1$.

If $y<0$, then $y<x-(x+y)y \leq 0$, which is impossible.

Hence, if $x+y=x\ /\ y$, then $(x=-4$ and $y=2)$ or $(x=-y-1$ and $y>0)$. Thus, $(A4)$ is a sufficient condition. On the other hand, if $(x=-4$ and $y=2)$ or $(x=-y-1$ and $y>0)$, then obviously $x+y=x\ /\ y$. Hence, $(A4)$ is necessary condition also.

$(A5)$ is the condition for test cases that distinguish $x+y$ from $x\ \%\ y$. First, any test case that satisfies the condition is sufficient to detect such a mutant. We need to consider three cases:

(i) $\boldsymbol{u*v>0}$

In this case, $(u>0$ and $v>0)$ or $(u<0$ and $v<0)$. Since u and v are all greater than zero, we have $u+v>v$ and $u\ \%\ v<v$. The case for $u<0$ and $v<0$ is similar. Hence, when $u*v>0$, $u+v \neq u\ \%\ v$.

(ii) $\boldsymbol{|u| \geq 2|v|}$

Thus, $u \geq 2|v|$ or $u \leq -2|v|$. If $u \geq 2|v|$, then $u+v \geq 2|v|+v \geq |v|$, while $u\ \%\ v<|v|$. If $u \leq 2|v|$, then $u+v \leq -2|v|+v<-|v|$, while $u\ \%\ v>-|v|$.

(iii) $\boldsymbol{|u|<|v|}$

In this case, $u\ \%\ v=u$ while $u+v \neq u$, since $v \neq 0$. On the other hand, if $(A5)$ cannot be satisfied, then $u*v<0$, $|u|<2|v|$, and $|u| \geq |v|$. If $u>0$ and $v<0$, then $-v \leq u<-2v$ and $u\ \%\ v=u+v$. If $u<0$ and $v>0$, then $-2v<u \leq -v$ and $u\ \%\ v=u+v$.

Hence, if $(A5)$ cannot be satisfied, the test cases cannot differentiate the mutant $x\ \%\ y$ from $x+y$.

$(A6)$ is the condition for distinguishing $x=y$ from $x+y$. The proof is obvious.

$(A7)$ is the condition for distinguish $x\ \&\&\ y$ from $x+y$. $(u+v \neq 1$ or $u=0$ or $v=0)$ and $(u \neq 0$ or $v \neq 0)$ implies $(u+v \neq 1$ or $u\ \&\&\ v=0)$ and

$(u+v \neq 0$ or $u \mathbin{\&\&} v = 1)$. In other words, $((u+v \neq 1$ and $(u+v \neq 0$ or $u \mathbin{\&\&} v = 1))$ or $((u \mathbin{\&\&} v = 0)$ and $(u+v \neq 0$ or $u \mathbin{\&\&} v = 1))$, so that $(u+v \neq 1$ and $u \mathbin{\&\&} v = 1)$ or $(u \mathbin{\&\&} v = 0$ and $u+v \neq 0)$. Obviously, $u+v \neq u \mathbin{\&\&} v$. If $(A7)$ is not satisfied, $(u+v = 1$ and $u \neq 0$ and $v \neq 0)$ or $(u = 0$ and $v = 0)$, then $(u+v = 1$ and $u \mathbin{\&\&} v = 1)$ or $(u+v = 0$ and $u \mathbin{\&\&} v = 0)$. Obviously, $u+v = v \mathbin{\&\&} v$.

Test cases that can satisfy $(A8)$ can distinguish $x \mathbin{||} y$ from $x+y$. Since $u \neq 0$ or $v \neq 0$, $u \mathbin{||} v = 1$. However, $u+v \neq 1$, and hence $u+v \neq u \mathbin{||} v$. If $(A8)$ is not satisfied, then $(u+v = 1)$ or $(u = 0$ and $v = 0)$. If $u+v = 1$, then $u+v = u \mathbin{||} v = 1$. If $u = v = 0$, then $u+v = u \mathbin{||} v = 0$. Hence, the mutant cannot be identified.

To differentiate any ORAN mutant from $x+y$, we need only find a test case that satisfies the conditions $(A1)$ to $(A8)$. For example, the test case $u = 5$ and $v = 9$ satisfies these conditions.

(*b*) $x - y$

Row 2 of Table 2 summaries the sufficient and necessary conditions for detecting the listed ORAN mutants of the expression $x - y$. The details are as follows:

$(B1)$:	$v \neq 0$
$(B2)$:	$u \neq v \;/\; (1-v)$
$(B3)$:	$u \neq v^2 \;/\; (v-1)$
$(B4)$:	$(u \neq v+1$ or $v < 0)$ and $(u \neq 4$ or $v \neq 2)$
$(B5)$:	$u * v < 0$, $\lvert u\rvert \geq 2\lvert v\rvert$ or $\lvert u\rvert < \lvert v\rvert$
$(B6)$:	$u \neq 2v$
$(B7)$:	$(u - v \neq 1$ or $u = 0$ or $v = 0)$ and $(u \neq 0$ or $v \neq 0)$
$(B8)$:	$u - v \neq 1$ and $(u \neq 0$ or $v \neq 0)$

$(B1)$, $(B2)$ and $(B3)$ are the conditions for test cases to detect the mutants $x+y$, $x*y$ and $x \;/\; y$ (float), respectively. Their proofs are obvious.

$(B4)$ is the condition for test cases to detect the mutant $x \;\%\; y$. The proof is similar to that of $(A4)$.

$(B5)$ is the condition for test case to detect the mutant $x \;/\; y$. If $x - y = x \;/\; y$, then $x = (x-y)y + n$. Hence, $n = x - (x-y)y$. If $x \geq 0$, then $0 \leq n < \lvert y\rvert$. If $x < 0$, then $-\lvert y\rvert < n \leq 0$. We have the following two cases:

(*i*) $x \geq 0$

If $y > 0$, then $y > x - (x-y)y \geq 0$. Hence, $y < x \leq y + 1 + 1 \;/\; (y-1)$. Thus, $x = y+1$ or $(x = 4$ and $y = 2)$.

If $y < 0$, it is impossible for $x - y = x \;/\; y$.

(*ii*) $x < 0$

When $x < 0$, we have $-\lvert y\rvert < x - (x-y)y \leq 0$. It is impossible for whatever value of y.

Obviously, $(B5)$ is a sufficient and necessary condition to detect the mutant $x \;/\; y$.

$(B6)$ is the condition for a test case to distinguish $x - y$ from $x = y$. If the test cases satisfy $u \neq 2v$, then $u - v \neq v$. In this case, $x - y$ can be distinguished from $x = y$. On the other hand, if $u = 2v$, then $u - v = v$. In this case, $x - y$ cannot be distinguished from $x = y$. Hence, a single test case cannot differentiate $x - y$ from $x = y$.

$(B7)$ and $(B8)$ are the conditions for test cases to detect the mutants $x \mathbin{\&\&} y$ and $x \mathbin{||} y$. Their proofs are similar to those of $(A7)$ and $(A8)$, respectively.

To differentiate any ORAN mutant from $x - y$, we need only find a test case that satisfies the conditions $(B1)$ to $(B8)$. For example, the test case $u = 5$ and $v = 9$ satisfies these conditions.

(*c*) $x * y$

Row 2 of Table 2 summaries the sufficient and necessary conditions for detecting the listed ORAN mutants of the expression $x * y$. The details are as follows:

$(C1)$:	$u \neq v \;/\; (v-1)$
$(C2)$:	$u \neq v \;/\; (1-v)$
$(C3)$:	$u \neq 0$ and $v \neq 1$ or -1
$(C4)$:	$u \neq 0$ and $v \neq 1$ or -1
$(C5)$:	$u \neq 0$
$(C6)$:	$u \neq 1$ and $v \neq 0$
$(C7)$:	$u * v \neq 1$ and $u * v \neq 0$
$(C8)$:	$u * v \neq 1$ and $(u \neq 0$ or $v \neq 0)$

The proofs of $(C1)$ and $(C2)$ are identical to those of $(A2)$ and $(B2)$, respectively.

$(C3)$ is obviously the condition to detect the mutant $x \;/\; y$ (float).

$(C4)$ is the condition for test cases to detect the mutant $x \;/\; y$ (int). If $x * y = x \;/\; y$ (int), then $x \geq xy^2$. Hence, $x = 0$ or $y = -1, +1$. In other words, $u = 0$ or $v = -1, +1$. On the other hand, if $u = 0$, then $x*y = x \;/\; y = 0$. If $v = -1, 1$, then $x*y = x \;/\; y = x = u$. Hence, $x*y = x \;/\; y$.

For $(C5)$, if $u = 0$, then $x*y = 0$ and $x \;\%\; y = 0$. Therefore, $x*y = x \;\%\; y$. On the other hand, if $x*y = x \;\%\; y$, then $\lvert x\rvert > \lvert x*y\rvert$ or $x = 0$. Since it is impossible to have $\lvert x\rvert > \lvert x*y\rvert$, we must have $x = 0$. Thus, $u = 0$.

$(C6)$ is the condition to detect the mutant $x = y$. If $u = 1$, then $x*y = u*v = v$. Since $x = y = v$, we have

$x * y$ identical to $x = y$. On the other hand, if $x + y$ is identical to $x = y$, then $x * y = y$. Hence, $x = 1$ or $y = 0$. Thus, $u = 1$ or $v = 0$.

The proofs of $(C7)$ and $(C8)$ are similar to those of $(A7)$ and $(A8)$, respectively.

To differentiate any ORAN mutant from $x * y$, we need only find a test case that satisfies the conditions $(C1)$ to $(C8)$. For example, the test case $u = 5$ and $v = 6$ satisfies these conditions.

***(d)* x / y (float)**

Row 2 of Table 2 summaries the sufficient and necessary conditions for detecting the listed ORAN mutants of the expression $x \ / \ y$ for floating point numbers. The details are as follows:

$(D1)$:	$u \neq v^2 \ / \ (1 - v)$
$(D2)$:	$u \neq v^2 \ / \ (v - 1)$
$(D3)$:	$u \neq 0$ and $v \neq 1$ or -1
$(D4)$:	$u \neq nv$ for any integer n
$(D5)$:	$u \neq 0$
$(D6)$:	$u \neq v^2$
$(D7)$:	$u \neq 0$ and $u \neq v$
$(D8)$:	$u \neq v$

The proofs of $(D1)$, $(D2)$ and $(D3)$ are identical to those of $(A3)$, $(B3)$ and $(C3)$, respectively.

$(D4)$ is used to detect the mutant $x \ / \ y$ (int). If $u = nv$ for some integer n, then $u \ / \ v$ (float) $= u \ / \ v$ (int) $= n$. On the other hand, if $(D4)$ is not true, then $|u \ / \ v \text{ (float)}| > |u \ / \ v \text{ (int)}|$. Thus, $(D4)$ is a sufficient and necessary condition.

The proof of $(D5)$ is similar to that of $(C5)$.

$(D6)$ is the condition for test cases to detect the mutant $x = y$. If $u = v^2$, then $u \ / \ v = v$. Therefore, $x \ / \ y$ has the same value as $x = y$. On the other hand, if $u \neq v^2$, then $u \ / \ v \neq v$. Hence, $x \ / \ y$ is not equal to $x = y$. Thus, $(D6)$ is a sufficient and necessary condition.

The proofs of $(D7)$ and $(D8)$ are similar to those of $(A7)$ and $(A8)$, respectively.

To differentiate any ORAN mutant from $x \ / \ y$ (float), we need only find a test case that satisfies the conditions $(D1)$ to $(D8)$. For example, the test case $u = 7$ and $v = 10$ satisfies these conditions.

***(e)* x / y (int)**

Row 2 of Table 2 summaries the sufficient and necessary conditions for detecting the listed ORAN mutants of the expression $x \ / \ y$ for integers. The details are as follows:

$(E1)$:	$(u \neq -v - 1$ or $v < 0)$ and $(u \neq -4$ or $v \neq 2)$
$(E2)$:	$(u \neq v + 1$ or $v > 0)$ and $(u \neq 4$ or $v \neq 2)$
$(E3)$:	$u \neq 0$ and $v \neq 1$ or -1
$(E4)$:	$u \neq nv$ for any integer n
$(E5)$:	$(u < 0$ or $|u| \neq n(v + 1))$ and $u \neq 0$
$(E6)$:	$u \neq v^2$
$(E7)$:	$u \ / \ v \neq 1$ and $u \neq 0$
$(E8)$:	$u \ / \ v \neq 1$

The proofs of $(E1)$, $(E2)$, $(E3)$ and $(E4)$ are identical to those of $(A4)$, $(B4)$, $(C4)$ and $(D4)$, respectively.

$(E5)$ is the condition to detect the mutant $x \ \% \ y$. If $v > 0$ and $|u| = n(v + 1)$, then $u \ / \ v = u \ \% \ v = n$. Hence, $x \ / \ y = x \ \% \ y$. On the other hand, suppose $(v < 0$ or $|u| \neq n(v + 1))$ and $u \neq 0$. If $u > 0$, then $u \ / \ v < 0$ and $u \ \% \ v > 0$. Hence, $u \ / \ v \neq u \ \% \ v$. If $u < 0$, then $u \ / \ v > 0$ and $u \ \% \ v < 0$. Also, $u \ / \ v \neq u \ \% \ v$. Thus, $(E5)$ is sufficient and necessary condition.

The proof of $(E6)$ is similar to that of $(D6)$.

The proofs of $(E7)$ and $(E8)$ are similar to those of $(A7)$ and $(A8)$, respectively.

To differentiate any ORAN mutant from $x \ / \ y$ (int), we need only find a test case that satisfies the conditions $(E1)$ to $(E8)$. For example, the test case $u = 2$ and $v = 5$ satisfies these conditions.

***(f)* x % y (int)**

Row 2 of Table 2 summaries the sufficient and necessary conditions for detecting the listed ORAN mutants of the expression $x \ \% \ y$ for integers. The details are as follows:

$(F1)$:	$u * v > 0$ or $|u| \geq 2|v|$ or $|u| < |v|$
$(F2)$:	$u * v < 0$, $|u| \geq 2|v|$ or $|u| < |v|$
$(F3)$:	$u \neq 0$
$(F4)$:	$u \neq nv$ for any integer n
$(F5)$:	$v < 0$ or $|u| \neq n(v + 1)$ and $u \neq 0$
$(F6)$:	No constraint
$(F7)$:	$u \neq v + 1$ and $u \neq 0$
$(F8)$:	$u \neq v + 1$

The proofs of $(F1)$, $(F2)$, $(F3)$, $(F4)$ and $(F5)$ are identical to those of $(A5)$, $(B5)$, $(C5)$, $(D5)$ and $(E5)$, respectively.

For $(F6)$, no constraint on test cases is necessary for distinguishing $x \ \% \ y$ from the mutant $x = y$, since $x \ \% \ y < y$.

The proofs of $(F7)$ and $(F8)$ are similar to those of $(A7)$ and $(A8)$, respectively.

To differentiate any ORAN mutant from $x \% y$, we need only find a test case that satisfies the conditions $(F1)$ to $(F8)$. For example, the test case $u = 20$ and $v = 3$ satisfies these conditions.

(g) $x = y$

Row 2 of Table 2 summaries the sufficient and necessary conditions for detecting the listed ORAN mutants of the expression $x = y$. The details are as follows:

$(G1)$:	$u \neq 0$
$(G2)$:	$u \neq 2v$
$(G3)$:	$u \neq 1$ and $v \neq 0$
$(G4)$:	$u \neq v^2$
$(G5)$:	$u \neq v^2$
$(G6)$:	No constraint
$(G7)$:	$(u = 0$ or $v \neq 1)$ and $v \neq 0$
$(G8)$:	$(u \neq 0$ or $v \neq 0)$ and $v \neq 1$

The proofs of $(G1)$, $(G2)$, $(G3)$, $(G4)$, $(G5)$ and $(G6)$ are identical to those of $(A6)$, $(B6)$, $(C6)$, $(D6)$, $(E6)$ and $(F6)$, respectively.

$(G7)$ is used to detect the mutant x && y. If $u = 0$ and $v \neq 0$, then u && $v = 0$ and $(u = v) \neq 0$. If $v \neq 1$ and $v \neq 0$, obviously u && $v \neq v$. On the other hand, suppose $(G5)$ is not true. In other words, $u \neq 0$ and $v = 1)$ or $v = 0$. If $u \neq 0$ and $v = 1$, then u && $v = v = 1$. If $v = 0$, then u && $v = v = 0$.

$(G8)$ is the condition to detect the mutant $x \mid\mid y$. If $u \neq 0$ and $v \neq 0$, then $u \mid\mid v = 1$. Since $v \neq 1$, we have $u \mid\mid v \neq v$. On the other hand, suppose $(G8)$ is not true. Then, $(u = 0$ and $v = 0)$ or $v = 1$. If $u = 0$ and $v = 0$, then $u \mid\mid v = v = 0$. If $v = 1$, then $u \mid\mid v = v = 1$.

To differentiate any ORAN mutant from $x = y$, we need only find a test case that satisfies the conditions $(F1)$ to $(F8)$. For example, the test case $u = 7$ and $v = 8$ satisfies these conditions.

For the case of a binary arithmetical operator being replacement by a relational operator, let E' denote a mutant of an atomic expression E with the arithmetic operator replaced by a relational operator. The sufficient and necessary condition for detecting such a mutant is $(E = 0$ and $E' \neq 0)$ or $(E \neq 0$ and $E' = 0)$. This condition can be added to the respective conditions in Table 2. No extra test case will be necessary.

For an expression with a unary operator, $x\odot$ or $\odot x$, its ORAN mutant is $x\odot'$ or $\odot' x$. A single test case is sufficient. If the operator is "++" or "−−", any value can be taken as a test case. If the operator is "+" or "−", any value except 0 can be taken as a test case. ■

If, in the domain of x and y, no values can distinguish the expression $x \odot y$ from the mutant $x \odot' y$, then we consider the mutant to be equivalent to the original.

5. Conclusion

In this paper, we have discussed in detail the testing of atomic arithmetic expressions. We have covered the fundamental mutants for such expressions, namely Uuor mutants, VDTR mutants and ORAN mutants. Contrary to the belief in code coverage methods, we find that a single test case is not sufficient for testing an atomic arithmetic expression. On the other hand, it is not as complex as other testers have thought. Two test cases are sufficient to detect all the fundamental single-fault mutants.

The results of this paper is not only useful for atomic arithmetic expressions but can be applied to further studies on the testing of complex programs.

References

[1] H. Agrawal, R. A. DeMillo, B. Hathaway, W. Hsu, W. Hsu, E. W. Krawser, R. J. Martin, A. Mathur and E. Spafford. Design of mutation operators for the C programming language. Technical Report SERC-TR-41p, Software Engineering Research Center, Department of Computer Science, Purdue University, Indiana, 1989.

[2] K. A. Foster. Error sensitive test cases analysis (ESTCA). *IEEE Transactions on Software Engineering*, 6 (3): 258–264, 1980.

[3] P. G. Frankl and E. J. Weyuker. Provable improvements on branch testing. *IEEE Transactions on Software Engineering*, 19 (10): 962–975, 1993.

[4] W. E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, 8 (4): 371–379, 1982.

[5] B. Jeng and E. J. Weyuker. A simplified domain-testing strategy. *ACM Transactions on Software Engineering and Methodology*, 3 (3): 254–270, 1994.

[6] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch and C. Zapf. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology*, 5 (2): 99–118, 1996.

[7] K.-C. Tai. Theory of fault-based predicate testing for computer programs. *IEEE Transactions on Software Engineering*, 22 (8): 552–562, 1996.

[8] J. M. Voas and G. McGraw. *Software Fault Injection: Inoculating Programs against Errors*, John Wiley, New York, 1998.

[9] L. J. White and E. I. Cohen. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering*, SE-6 (3): 247–257, 1980.