EXAMPLE 5.9

Calculate the cyclomatic number using the invalid control flow graph shown in Fig. 5-4.

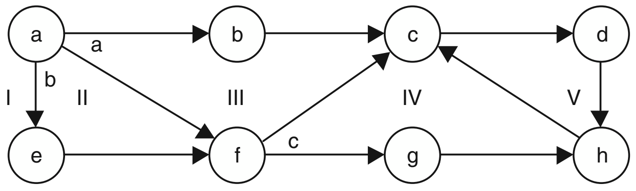

Fig. 5-4. Invalid control flow graph.

The cfg is the same as earlier examples, except that the c-h arc has been replaced by an h-c arc. This will not change the counts of nodes, edges, or regions. Thus, the first two methods of counting the cyclomatic number will not change. However, decision d has been eliminated, so the third method will not give the same answer. However, this is not a valid cfg, since there is now no stopping node.

Threshold Value

An important aspect of a metric is guidance about when the values are reasonable and when the values are not reasonable. McCabe analyzed a large project and discovered that for modules with cyclomatic number over 10, the modules had histories of many more errors and many more difficulties in maintenance. Thus, 10 has been accepted as the threshold value for the cyclomatic number in a module. If the cyclomatic number is greater than 10, efforts should be made to reduce the value or to split the module.

5.3.2 HALSTEAD'S SOFTWARE SCIENCE

Maurice Halstead was one of the first researchers in software metrics. He did his work in the late 1960s and 1970s. His goal was to identify what contributed to the complexity in software. He empirically looked for measures of intrinsic size. After finding what he felt were good measures and prediction formulas, he tried to develop a coherent theory. His simple metrics are still considered valid, while his more complex metrics and prediction formulas have been found to be suspect.

Basic Entities—Operators and Operands

The basic approach that gave Halstead good results was to consider any program to be a collection of tokens, which he classified as either operators or operands. ***Operands*** were tokens that had a value. Typically, variables and constants were operands. Everything else was considered an ***operator***. Thus, commas, parentheses, arithmetic operators, brackets, and so forth were all considered operators.

All tokens that always appear as a pair, triple, and so on will be counted together as one token. For example, a left parenthesis and a right parenthesis will be considered as one occurrence of the token parenthesis. A language that has an if-then construction will be considered to have an if-then token.

Halstead also was concerned about algorithms and not about declarations, i/o statements, and so on. Thus, he did not count declarations, input or output statements, or comments. However, currently most organizations would count all parts of a program.

Halstead's definitions of operator and operand are open to many interpretations. No standard has been accepted for deciding ambiguous situations. The good news is that as long as an organization is consistent, it doesn't matter. The bad news is that people in one organization cannot compare their results with the results from other organizations.

The author recommends a syntax-based approach where all operands are user-defined tokens and all operators are the tokens defined by the syntax of the language.

Basic Measures—η_1 and η_2

The count of unique operators in a program is η_1 (pronounced "eta one"), and the count of unique operands in a program is η_2 (pronounced "eta two").

The total count of unique tokens is $\eta = \eta_1 + \eta_2$. This is the basic measure of the size of the program.

EXAMPLE 5.10

Identify the unique operators and operands in the following code that does multiplication by repeated addition.

```
Z = 0;
while X > 0
      Z = Z + Y ;
      X = X-1 ;
end-while ;
print(Z) ;
```

operators

```
= ; while-endwhile > +-print ()
```

operands

```
Z 0 X Y 1
```

thus, η_1 = 8 and η_2 = 5

Potential Operands, η_2*

Halstead wanted to consider and compare different implementations of algorithms. He developed the concept of potential operands that represents the minimal set of values needed for any implementation of the given algorithm. This is usually calculated by counting all the values that are not initially set within the algorithm. It will include values read in, parameters passed in, and global values accessed within the algorithm.

Length, N

The next basic measure is total count of operators, N_1, and the total count of operands, N_2. These are summed to get the length of the program in tokens:

$$N = N_1 + N_2$$

EXAMPLE 5.11
Calculate Halstead's length for the code of Example 5.10.

operators
```
=              3
;              5
while-endwhile 1
>              1
+              1
-              1
print               1
()             1
```

operands
```
Z        4
0        2
X        3
Y        2
1        1
```

There are 14 occurrences of operators, so N_1 is 14. Similarly, N_2 is 12.
$N = N_1 + N_2 = 14 + 12 = 26$.

Estimate of the Length (est N or N_hat)

The estimate of length is the most basic of Halstead's prediction formulas. Based on just an estimate of the number of operators and operands that will be used in a program, this formula allows an estimate of the actual size of the program in terms of tokens:

$$\text{est N} = \eta_1 * \log_2 \eta_1 + \eta_2 * \log_2 \eta_2$$

EXAMPLE 5.12
Calculate the estimated length for the code of Example 5.10.
The $\log_2$ of *x* is the exponent to which 2 must be raised to give a result equal to *x*. So, log2 of 2 is 1, log2 of 4 is 2, of 8 is 3, of 16 is 4:

$$\log_2 \text{ of } \eta_1 = \log_2 8 = 3$$
$$\log_2 \text{ of } \eta_2 = \log_2 5 = 2.32$$
$$\text{est N} = 8 * 3 + 5 * 2.32 = 24 + 11.6 = 35.6$$

while the actual N is 26. This would be considered borderline. It is probably not a bad approximation for such a small program.

From experience, I have found that if N and est N are not within about 30 percent of each other, it may not be reasonable to apply any of the other software science measures.

Volume, V

Halstead thought of volume as a 3D measure, when it is really related to the number of bits it would take to encode the program being measured.[4] In other words:

$$V = N * \log_2(\eta_1 + \eta_2)$$

EXAMPLE 5.13

Calculate V for the code of Example 5.10.

$$V = 26 * \log_2 13 = 26 * 3.7 = 96.2$$

The volume gives the number of bits necessary to encode that many different values. This number is hard to interpret.

Potential Volume, V*

The potential volume is the minimal size of a solution to the problem, solved in any language. Halstead assumes that in the minimal implementation, there would only be two operators: the name of the function and a grouping operator. The minimal number of operands is $\eta_2{}^*$.

$$V^* = (2 + \eta_2^*) \log_2(2 + \eta_2^*)$$

Implementation Level, L

Since we have the actual volume and the minimal volume, it is natural to take a ratio. Halstead divides the potential volume by the actual. This relates to how close the current implementation is to the minimal implementation as measured by the potential volume. The implementation level is unitless.

$$L = V^*/V$$

The basic measures described so far are reasonable. Many of the ideas of operands and operators have been used in many other metric efforts. The remaining measures are given for historical interest and are not recommended as being useful or valid.

Effort, E

Halstead wanted to estimate how much time (effort) was needed to implement this algorithm. He used a notion of elementary mental discriminations (emd).

$$E = V/L$$

The units are elementary mental discriminations (emd). Halstead's effort is not monotonic—in other words, there are programs such that if you add statements, the calculated effort decreases.

[4] Encoding *n* different items would require at a minimum $\log_2 n$ bits for each item. To encode a sequence of N, such items would require $N * \log_2 n$.

Time, T

Next, Halstead wanted to estimate the time necessary to implement the algorithm. He used some work developed by a psychologist in the 1950s, John Stroud. Stroud had measured how fast a subject could view items passed rapidly in front of his face. S is the Stroud number (emd/sec) taken from those experiments. Halstead used 18 emd/sec as the value of S.

$$T = E/S$$

5.3.3 HENRY–KAFURA INFORMATION FLOW

Sallie Henry and Dennis Kafura developed a metric to measure the intermodule complexity of source code. The complexity is based on the flow of information into and out of a module. For each module, a count is made of all the information flows into the module, in_i, and all the information flows out of the module, out_i. These information flows include parameter passing, global variables, and inputs and outputs. They also use a measure of the size of each module as a multiplicative factor. LOC and complexity measures have been used as this weight.

$$HK_i = weight_i * (out_i * in_i)^2$$

The total measure is the sum of the HK_i from each module.

EXAMPLE 5.14

Calculate the HK information flow metrics from the following information. Assume the weight of each module is 1.

mod #	a	b	c	d	e	f	g	h
in_i	4	3	1	5	2	5	6	1
out_i	3	3	4	3	4	4	2	6

mod	a	b	c	d	e	f	g	h
HK_i	144	81	16	225	64	400	144	36

HK for the whole program will be 1110.