A non algebraic example of a constructive correctness proof.

We consider a text built from five types of characters, viz. digits, operators, "(", ")" and ";". The text can be read from left to right by means of the primitive "move" which assigns to the variable named "x" the next character of the text. Initially the value of x is undefined, after the first execution of "move", x equals the left-most character of the text. The text has infinite length and it is required to write a recognizer called "sent" that has to establish whether the text starts with a $<$ sent $>$, given by the following syntax.

$$
\begin{aligned}
&< \text{sent} > ::= < \text{exp} > ; \\
&< \text{exp} > ::= < \text{term} > \{< \text{operator} >< \text{term} >\} \\
&< \text{term} > ::= < \text{digit} > \{< \text{digit} >\} \mid (< \text{exp} >)
\end{aligned} \qquad (1)
$$

Here $\{ \ldots\ldots \}$ should be read as "zero or more times the enclosed". An alternative syntax would have been

$$
\begin{aligned}
&< \text{sent} > ::= < \text{exp} > ; \\
&< \text{exp} > ::= \{< \text{term} >< \text{operator} >\} < \text{term} > \\
&< \text{term} > ::= \{< \text{digit} >\} < \text{digit} > \mid (< \text{exp} >)
\end{aligned} \qquad (2)
$$

We regard (1) and (2) as trivially equivalent.

In order to carry out this investigation, the routine has at its disposal five boolean functions, defined when x has a value, viz. digit(x) operator(x), open(x), close(x) and semi(x), such that always exactly one of the five has the value true.

The result of the investigation has to be recorded in the global boolean named "c" (short for: correct). In order to specify more rigorously the net effect to be established by a call on "sent", we introduce the following notations.

Let S be the non-empty string of characters read by "sent"; by definition x then equals the right-most character of S and by $S - x$ we denote the string of characters of S up to and excluding this right-most character.

We now want to express two things: if a < sent > is found, S should equal that sentence, if, however, sent has established that the text does not start with a < sent >, no more characters than are needed for this conclusion should have been read. In view of our later needs, we introduce two new syntactic categories, viz. < pbo sent > and < bo sent >. Here < pbo sent > stands for "proper begin of a sentence but not a complete sentence", i.e. a string of characters not forming a sentence all by itself, but that still can be extended to form a correct sentence; < bo sent >, being short for "begin of sentence", is either a correct sentence all by itself, or allowing extension to a correct sentence. In other words

$$< \text{bo sent} > ::= < \text{pbo sent} > \mid < \text{sent} > \tag{3}$$

Similarly we introduce

$$< \text{bo exp} > ::= < \text{pbo exp} > \mid < \text{exp} > \tag{4}$$

$$< \text{bo term} > ::= < \text{pbo term} > \mid < \text{term} > \tag{5}$$

Note: A < pbo Y > has <u>not</u> been defined as a string that by a non-empty extension can be transformed into a < Y > : it has also been required not to be a < Y > all by itself.

In terms of S and the notion < pbo sent > we now specify the desired net effect of "sent" by requiring that upon completion

$$\begin{aligned} &S - x = < \text{pbo sent} > \underline{\text{and}} \\ &\quad S \neq < \text{pbo sent} > \underline{\text{and}} \\ &\quad c = [\ S = < \text{sent} > \ ] \end{aligned} \tag{6}$$

The first of these terms is required to ensure that not too much has been read, the second guarantees that enough has been read and the final term defines the value of c. Note that specification (6) has been given without taking syntax (1) into account! <u>All</u> proper initial strings of S are of the syntactical category < pbo sent > : the empty string is, and as soon as reading has progressed such that the first two terms are true, reading stops and c will get its proper value.

In order to proceed with (6) in which that syntactic category < pbo sent > occurs, we derive from the first line of (1)

$$< \text{pbo sent} > ::= < \text{bo exp} > \tag{7}$$

where $\langle$ bo exp $\rangle$ comprises all expressions and sequences that can be extended to an expression.

(Note: According to (4) it is not excluded that a proper initial string of an expression by itself is an expression!)

Relation (7), together with the first line of the syntax (1) enables us to rewrite the desired net effect of "sent" as follows: upon termination of "sent" we require

$$\begin{aligned} & S - x = \langle \text{bo exp} \rangle \ \underline{\text{and}} \\ & \quad S \neq \langle \text{bo exp} \rangle \ \underline{\text{and}} \\ & \quad c = [\ S - x = \langle \text{exp} \rangle \ \underline{\text{and}}\ \text{semi}(x)\ ] \end{aligned} \tag{8}$$

The second line shows that the concept $\langle$ bo exp $\rangle$ is related to the complete string S. As a result, when we wish to express "sent" in terms of a primitive "exp", whose analysis has to embody the syntactical rules of an $\langle$ exp $\rangle$ and that shall read a string E, then E and S should be identical.

In terms of a primitive "exp", reading a string E, such that upon termination

$$\begin{aligned} & E - x = \langle \text{bo exp} \rangle \ \underline{\text{and}} \\ & \quad E \neq \langle \text{bo exp} \rangle \ \underline{\text{and}} \\ & \quad c = [\ E - x = \langle \text{exp} \rangle\ ] \end{aligned} \tag{9}$$

the body

$$\underline{\text{proc}}\ \text{sent:}\ \ \text{exp};\ c := c\ \underline{\text{and}}\ \text{semi}(x)\ \underline{\text{corp}} \tag{10}$$

satisfies requirement (8). A formal proof of the correctness of (10) relies on the axiom of assignment only: with post-condition (8) we derive on account of (1) as the post-condition for its call of "exp"

$$\begin{aligned} & S - x = \langle \text{bo exp} \rangle \ \underline{\text{and}} \\ & \quad S \neq \langle \text{bo exp} \rangle \ \underline{\text{and}} \\ & c\ \underline{\text{and}}\ \text{semi}(x) = [\ S - x = \langle \text{exp} \rangle\ \underline{\text{and}}\ \text{semi}(x)] \end{aligned}$$

which is obviously satisfied by E = S and specification (9). The equality E = S is guaranteed when, prior to its call on "exp", "sent" will not command any "move". It does not: it opens with "exp".

Now we are left with the duty of constructing a recognizer "exp", reading

a string E given by the first two lines of (9) and assigning to c the value given by the last line of (9). Because (9) mentions the syntactic category $<$ bo exp $>$, we had better derive its syntactic definition, now from the second line of the syntax. Syntax (2) is the most convenient and we find -because $<$ operator $>$ is always a single character! -

$$< \text{bo exp} > ::= \{< \text{term} \times \text{operator} >\} < \text{bo term} > \qquad (11)$$

The braces -"zero or more times"- suggest a while loop and therefore we must try to find a convenient invariant, implied by (9). It is certainly necessary to maintain

$$E - x = < \text{bo exp} >$$

as this takes care of the requirement that not too much will be read. On account of (11) we can then write

$$E = \{< \text{term} \times \text{operator} >\}\ T'$$
$$\text{with} \qquad T' - x = < \text{bo term} > \qquad (12)$$

Equation (11) tells us that the necessary and sufficient condition for $E = < \text{bo exp} >$ is in terms of T', as given by (12),

$$T' = < \text{bo term} > \underline{\text{or}}\ T' = < \text{term} \times \text{operator} > .$$

The necessary and sufficient condition for $E \neq < \text{bo exp} >$ is therefore

$$T' \neq < \text{bo term} > \underline{\text{and}}\ T' \neq < \text{term} \times \text{operator} >$$

or

$$T' \neq < \text{bo term} > \underline{\text{and non}}\ [\ T' - x = < \text{term} > \underline{\text{and}}\ \text{operator}(x)]$$

Furthermore is follows from (12) and the syntax for $< \text{exp} >$ that $E - x = < \text{exp} >$ (from the third line of (9)) is equivalent to $T' - x = < \text{term} >$. Therefore we can rewrite (9) as

$$Q\ \underline{\text{and non}}\ [\ c\ \underline{\text{and}}\ \text{operator}(x)]$$

with Q:

$$\begin{aligned} E - x &= < \text{bo exp} > \underline{\text{and}} \\ T' - x &= < \text{bo term} > \underline{\text{and}} \\ T' &\neq < \text{bo term} > \underline{\text{and}} \\ c &= [\ T' - x = < \text{term} >] \end{aligned} \qquad (13)$$

where we have combined in Q all assertions about the total text read by "exp". Now the invariance theorem for loops suggests a body for "exp" of the form

establish Q:

while c **and** operator(x) **do** something without violating Q **od**

Analogous to (9) we try a **primitive** "term", such that upon completion the string T read by it and c satisfy

$$T - x = \langle \text{bo term} \rangle \text{ and}$$
$$T \neq \langle \text{bo term} \rangle \text{ and}$$
$$c = [\, T - x = \langle \text{term} \rangle] \qquad (14)$$

Relations (12), (13) and (14) suggest that we try to prove that

$$E = \{\langle \text{term} \rangle \langle \text{operator} \rangle\} \qquad (15)$$

is a sufficient precondition for "term" to ensure the postcondition Q. As far as the effect on E of the primitive "term" is concerned, an execution of "term" is equivalent to

$$E := E + T$$

(where we have used the "+" for concatenation) or

$$E - x := E + T - x$$

The post-condition $E - x = \langle \text{bo exp} \rangle$ reduces thanks to the axiom of assignment and (11) to the pre-condition

$$E + T - x = \{\langle \text{term} \rangle \langle \text{operator} \rangle\} \langle \text{bo term} \rangle \; .$$

Because $T - x = \langle \text{bo term} \rangle$, (15) is a sufficient precondition, allowing the identification T' = T, guaranteeing on account of (14) the last three lines of Q as well. Because $E = \langle \text{empty} \rangle$ is a specific instance of (15), the initialization of the loop -i.e. establishing Q to start with- can be done by a single call on "term". As repeatable statement "term" would be acceptable, provided that we can show that

Q **and** c **and** operator(x) => relation (15)

The left-hand implies

$$E - x = \{\langle \text{term} \rangle \langle \text{operator} \rangle\} T' - x \text{ and}$$
$$T' - x = \langle \text{term} \rangle \text{ and}$$
$$\text{operator}(x)$$

and therefore (15) holds, but (15) was a sufficient pre-condition for "term" not to violate Q and we have found the program

proc exp: term; while c and operator(x) do term od corp .

We are left with the duty to construct a primitive "term" such that the string T, read by it, and c will satisfy (14) upon completion. In order to construct it, we derive from the third line of syntax (2) the syntactic rule

$$< \text{bo term} > ::= \{< \text{digit} >\} \mid (< \text{bo exp} > \mid (< \text{exp} >) \qquad (16)$$

and the primary duty of "term" is to establish a string T such that

$$T - x = < \text{bo term} > \text{ and}$$
$$T \neq < \text{bo term} > .$$

According to the syntax (16) we have three cases

A) $T - x = \{< \text{digit} >\}$

In this case $x \neq < \text{digit} >$, on account of the requirement $T \neq < \text{bo term} >$, and c is to get the value true when $T - x \neq < \text{empty} >$.

B) $T - x = (< \text{bo exp} >$

or, with $T = (E \qquad (17)$

we then know that

$$E - x = < \text{bo exp} > \text{ and } E \neq < \text{bo exp} > . \qquad (18)$$

But we know also

$$\text{non } [\, E - x = < \text{exp} > \text{ and close}(x)\,] \qquad (19),$$

because otherwise $T = < \text{bo term} >$ (3rd alternative) would have been true. In this case c must get the value false, because from $T - x = < \text{term} >$ would follow $E - x = < \text{exp} >$) and this is incompatible with $E - x = < \text{bo exp} >$ because via mathematical induction it is easily proved that in an $< \text{exp} >$ the number of opening brackets equals the number of closing brackets while in a $< \text{bo exp} >$ the number of closing brackets does not exceed the number of opening brackets.

C) $T - x = (< \text{exp} >)$

In this case, regardless the value of x, $T \neq <$bo term$>$ holds and c must get the value true. With

$$T - x = (E$$

-in contrast to (17), covering case B- we have again relation (18) but also

$$E - x = < \text{exp} > \text{ and close}(x).$$

But this is exactly the negation of relation (19)! Thus we are led to the following program (with comments inserted between braces)

```
proc term:   move;
             if open(x)
               then {case B and case C} exp;
                    if c and close(x)
                      then {case C} move
                      else {case B} c:= false
                    fi
               else {case A} c:= digit(x);
                    while digit(x) do move od
             fi
corp
```

* * *

Conclusion. The primary purpose of the above exercise was to demonstrate that the applicability of the technique of developing correctness proof and program hand in hand is not confined to simple programs dealing with algebraic relations between integers but can also be applied to non-arithmetic data processing. Some readers might argue that this example rather weakens my point, because the essay is rather long compared to the length of the program. My answer to them is the following:

1) A program need not be long in order to be difficult to conceive and also, unless one has produced absolutely insipid code, a final program is always a a very compact product of one's intellectual labour.

2) We have not made strong assumptions about the reader's familiarity with the theory of recursively defined syntaxes and therefore have derived explicitly all we needed.

3) Anyone who thinks that he can make reliable programs without a sufficient "theory" about the subject matter of the computations, fools himself.

Finally, although the alert reader will have noticed it, it seems worthwhile to draw attention to the fact, that in all our reasoning in the above exercise we have not given a single example of a sentence, nor of a non-sentence.

Eindhoven, 18th May 1973 Edsger W.Dijkstra