

A more formal treatment of a less simple example.

For obvious reasons, most programming experiments that have been carried out in the exploration of formal techniques, dealt with simple, algebraic examples. For equally obvious reasons, the examples shown in tutorial texts on this subject are mostly of the same nature. (There has been a time when all of Computing Science seemed to boil down to massaging Euclid's Algorithm for the greatest common divisor!) This paper is primarily directed at remedying this situation.

* * *

Our ultimate goal is to develop a program that will transform expressions from infix notation to postfix notation. The subject matter to be manipulated by our program are therefore not integers, but strings of characters that may, or may not belong to certain syntactic categories. For variables of type "character string" we have to have at our disposal the analogon of high-school algebra (such as $(a > b \text{ and } c > d) \Rightarrow a + c > b + d$, etc.) that sufficed for the well-known numerical examples. Before embarking on our problem proper, we shall first introduce the necessary formal apparatus and the notation needed for its description.

We assume our syntax given in BNF. Let $\langle pqr \rangle$ denote a syntactical category. We shall then express the fact that a string named K belongs to the syntactical category $\langle pqr \rangle$ by

$$pqr(K) \quad .$$

For strings (named K, L, \dots) and characters (named y, z, \dots) we shall denote concatenation by juxtaposition, e.g. $KL, Ky, Ky;$ etc. If L may be any string and y may be any character, any non-empty string may be denoted by yL or Ly .

With any syntactic category $\langle pqr \rangle$ we may associate the syntactic category $\langle bopqr \rangle$ --"begin of a $\langle pqr \rangle$ "-- consisting of all the strings that either are a $\langle pqr \rangle$ or can be extended at the right-hand side so as to become a $\langle pqr \rangle$ or both. According to that definition the statement that the syntactic category $\langle pqr \rangle$ is not empty --i.e. contains, as most useful syntactic categories, at least one string-- is equivalent with the predicate

$$bopqr(\text{empty string}) \quad .$$

The formal definition of the predicate `bopqr` in terms of `pqr` --with `K` and `L` denoting arbitrary strings-- is

$$\text{bopqr}(K) \Leftrightarrow (\exists L: \text{pqr}(KL)) \quad (1)$$

Separating the case that `L` is empty and the case that `L` is not empty, we can rewrite (1) as

$$\text{bopqr}(K) \Leftrightarrow (\text{pqr}(K) \text{ or } (\exists yL: \text{pqr}(KyL)))$$

which, thanks to (1), can be reduced to

$$\text{bopqr}(K) \Leftrightarrow (\text{pqr}(K) \text{ or } (\exists y: \text{bopqr}(Ky))) \quad (2)$$

from which we immediately derive

$$(\text{bopqr}(K) \text{ and } (\forall y: \text{non bopqr}(Ky))) \Rightarrow \text{pqr}(K) \quad (3)$$

From (1) we derive further

$$\begin{aligned} \text{bopqr}(Ky) &\Leftrightarrow (\exists L: \text{pqr}(KyL)) \\ &= (\exists yL: \text{pqr}(KyL)) \\ &\Rightarrow \text{bopqr}(K) \end{aligned}$$

From this result

$$\text{bopqr}(Ky) \Rightarrow \text{bopqr}(K) \quad (4)$$

follows that $\langle \text{bopqr} \rangle = \langle \text{bobopqr} \rangle$.

Because $\text{pqr}(K) \Rightarrow (\exists L: \text{pqr}(KL))$ --`L` = the emptystring does the job-- a further consequence of (1) is

$$\text{pqr}(K) \Rightarrow \text{bopqr}(K) \quad (5)$$

From our informal description of what we intended the notion "begin of" to mean, the above is all intuitively obvious, and by now the reader may wonder what all the fuss is about. The point is that we need such formulae as soon as we wish to give a more rigorous treatment of a parser.

* * *

We intend to develop a mechanism called "sentsearch" that is intended to recognize strings from the syntactical category $\langle \text{sent} \rangle$. More precisely, we assume that the input string can be scanned in the order from left to right and reserve the identifier "x" for the next visible character of the input string. If the input string starts with "a + b" , then we have initially

$x = "a"$; after the execution of "move" the relation $x = "+"$ will hold. Besides assigning a new value to x , the primitive "move" can be viewed as also appending the old value of x to the right to "the strings of characters moved over" or "the string of characters read" or "the string of characters that are no longer visible."

Let S be the string of characters "moved over" by an activation of `sentsearch` .

Note 1. When developing the body of `sentsearch` we may assume that a local so-called "ghost variable" S is initialized at the beginning as the empty string, that each call on "move" is implicitly preceded by " $S := Sx$ " , and that upon termination S is handed back as a "ghost function value" to the calling environment. (End of Note 1.)

In the case that the input sequence does not start with a $\langle \text{sent} \rangle$, we want S to be the sequence that is insufficient to establish this fact, while Sx is long enough to make this conclusion. That is, upon termination

$$\text{bosent}(S) \text{ and } \underline{\text{non}} \text{ bosent}(Sx)$$

will hold. The first term expresses that not too much has been moved over, the second term expresses that enough has been moved over. In the case that the input string does start with a $\langle \text{sent} \rangle$, we wish S to be equal to that $\langle \text{sent} \rangle$ and assume our syntax for $\langle \text{sent} \rangle$ --about which nothing has been given yet-- to satisfy

$$\text{sent}(L) \Rightarrow \underline{\text{non}} (\exists y: \text{bosent}(Ly)) \quad (6)$$

Whether or not a $\langle \text{sent} \rangle$ has been found is to be recorded in the global boolean c --short for "correct"-- and our complete specification of `sentsearch` is that it has to establish $R_s(S, x, c)$; where $R_s(S, x, c)$ is given by

$$\text{bosent}(S) \text{ and } \underline{\text{non}} \text{ bosent}(Sx) \text{ and } c = \text{sent}(S) \quad (7)$$

Note 2. The consequence of assumption (6) is that when the input string starts with a $\langle \text{sent} \rangle$ and the analysis has progressed to S equal to that $\langle \text{sent} \rangle$, the term $\underline{\text{non}} \text{ bosent}(Sx)$ is true for all possible values of x , i.e. `sentsearch` can then terminate without inspecting the next visible character. The end of a $\langle \text{sent} \rangle$ is assumed to be detectable without looking beyond it. (End of Note 2.)

We now give the syntax for $\langle \text{sent} \rangle$:

$$\langle \text{sent} \rangle ::= \langle \text{exp} \rangle ; \quad (8)$$

From this we have to derive the syntax for the syntactical category $\langle \text{bosent} \rangle$:

$$\langle \text{bosent} \rangle ::= \langle \text{sent} \rangle \mid \langle \text{boexp} \rangle \quad (9)$$

Each $\langle \text{bosent} \rangle$ can be derived by taking a $\langle \text{sent} \rangle$ and removing at the right-hand side zero or more characters from it. Removal of zero characters gives the first alternative, removal of one or more characters from " $\langle \text{exp} \rangle ;$ " boils down --because the semicolon is a single character-- to the removal of zero or more characters from $\langle \text{exp} \rangle$: but that is by definition the syntactic category called $\langle \text{boexp} \rangle$. Hence (9). The two alternatives are mutually exclusive, for we have for any string L :

$$\text{boexp}(L) \Rightarrow \underline{\text{non}} \text{sent}(L) \quad (10)$$

This can be proved by deriving a contradiction from $\text{boexp}(L)$ and $\text{sent}(L)$.
From $\text{boexp}(L)$ follows --according to (2)--

$$\text{exp}(L) \underline{\text{or}} (\exists y: \text{boexp}(Ly))$$

We deal with both term separately:

$$\text{exp}(L) \Rightarrow (\text{on account of (8)})$$

$$\text{sent}(L;) \Rightarrow (\text{on account of (5)})$$

$$\text{bosent}(L;) \Rightarrow (\exists y: \text{bosent}(Ly)) \quad ;$$

the second term gives

$$(\exists y: \text{boexp}(Ly)) \Rightarrow (\text{on account of (9)})$$

$$(\exists y: \text{bosent}(Ly)) \quad .$$

As both terms of the disjunction imply the same, we conclude that also

$$\text{boexp}(L) \Rightarrow (\exists y: \text{bosent}(Ly)) \quad .$$

According to (6), however,

$$\text{sent}(L) \Rightarrow \underline{\text{non}} (\exists y: \text{bosent}(Ly)) \quad .$$

The desired contradiction has been established and (10) has been proved.

Syntax rule (8) strongly suggests that the body of `sentsearch` should start with a call of `expsearch`. In order to design `sentsearch` in terms of `expsearch` we only need to know the net effect of `expsearch` and we propose in analogy to (7) that --when E is the string of characters moved over by `expsearch`-- the primitive `expsearch` will establish $\text{Re}(E, x, c)$, where $\text{Re}(E, x, c)$ is given by

$$\text{boexp}(E) \text{ and } \underline{\text{non}} \text{ boexp}(Ex) \text{ and } c = \text{exp}(E) \quad (11)$$

Designing `sentsearch` in terms of `expsearch` means that we would like to have theorems, such that from the truth of a relation of the form `Re` the truth of relations of the form `Rs` can be concluded. There are three such theorems.

Theorem 1. $(\text{Re}(L, x, c) \text{ and } \underline{\text{non}} c) \Rightarrow \text{Rs}(L, x, c)$

Proof. Assumed:

0. $\text{Re}(L, x, c) \text{ and } \underline{\text{non}} c$
 Derived:
1. $\text{boexp}(L)$ with (11) from 0
 2. $\text{bosent}(L)$ with (9) from 1
 3. $c = \text{exp}(L)$ with (11) from 0
 4. $\underline{\text{non}} c$ from 0
 5. $\underline{\text{non}} \text{exp}(L)$ from 3 and 4
 6. $\underline{\text{non}} \text{sent}(Lx)$ with (8) from 5
 7. $\underline{\text{non}} \text{boexp}(Lx)$ with (11) from 0
 8. $\underline{\text{non}} \text{bosent}(Lx)$ with (9) from 6 and 7
 9. $\underline{\text{non}} \text{sent}(L)$ with (10) from 1
 10. $c = \text{sent}(L)$ from 4 and 9
 11. $\text{Rs}(L, x, c)$ with (7) from 2, 8 and 10
- (End of Proof of Theorem 1.)

Theorem 2. $(\text{Re}(L, x, c) \text{ and } c \text{ and } \underline{\text{non}} \text{semi}(x)) \Rightarrow \text{Rs}(L, x, \text{false})$

Proof. Assumed:

0. $\text{Re}(L, x, c) \text{ and } c \text{ and } \underline{\text{non}} \text{semi}(x)$
 Derived:
1. $\text{boexp}(L)$ with (11) from 0
 2. $\text{bosent}(L)$ with (9) from 1
 3. $\underline{\text{non}} \text{semi}(x)$ from 0
 4. $\underline{\text{non}} \text{sent}(Lx)$ with (8) from 3
 5. $\underline{\text{non}} \text{boexp}(Lx)$ with (11) from 0
 6. $\underline{\text{non}} \text{bosent}(Lx)$ with (9) from 4 and 5
 7. $\text{false} = \text{sent}(L)$ with (10) from 1
 8. $\text{Rs}(L, x, \text{false})$ with (7) from 2, 6 and 7
- (End of Proof of Theorem 2.)

Theorem 3. $(\text{Re}(L, x, c) \text{ and } c \text{ and } \text{semi}(x)) \Rightarrow \text{Rs}(Lx, y, c)$

Proof. Assumed:

0. $\text{Re}(L, x, c) \text{ and } c \text{ and } \text{semi}(x)$
 Derived:
1. $c = \text{exp}(L)$ with (11) from 0
 2. c from 0
 3. $\text{exp}(L)$ from 1 and 2
 4. $\text{semi}(x)$ from 0
 5. $\text{sent}(Lx)$ with (8) from 3 and 4
 6. $c = \text{sent}(Lx)$ from 2 and ~~X5~~
 7. $\text{bosent}(Lx)$ with (5) from 5
 8. non $\text{bosent}(Lxy)$ with (6) from 5
 9. $\text{Rs}(Lx, y, c)$ with (7) from 7, 8 and 6.
- (End of Proof of Theorem 3.)

And now a possible body of `sentsearch` is evident, when we realize that its call on `expsearch` implies for the ghost variable `S` the assignment "`S := SE`"

```
proc sentsearch: { S = empty string}
    expsearch {Re(S, x, c)};
    if non c → skip
    || c and non semi(x) → c := false
    || c and semi(x) → move
    fi {Rs(S, x, c)}
```

corp

Note 3. Instead of Theorems 1 and 2 we could have discovered
 Theorem 1'. $(\text{Re}(L, x, c) \text{ and } \text{non } c) \Rightarrow \text{Rs}(L, x, \text{false})$
 Theorem 2'. $(\text{Re}(L, x, c) \text{ and } \text{non } \text{semi}(x)) \Rightarrow \text{Rs}(L, x, \text{false})$.
 This would have directed us towards the design of the body

```
proc sentsearch: expsearch;
    if non c or non semi(x) → c := false
    || c and semi(x) → move
    fi
```

corp

which, thanks to de Morgan's Theorem, has no aborting alternative construct.
 (End of note 3.)

We now consider for $\langle \text{exp} \rangle$ the following syntax

$$\langle \text{exp} \rangle ::= \langle \text{adder} \rangle \langle \text{term} \rangle \quad (12)$$

$$\langle \text{adder} \rangle ::= \{ \langle \text{term} \rangle \langle \text{adop} \rangle \} \quad (13)$$

$$\langle \text{adop} \rangle ::= + \mid - \quad (14)$$

where the braces indicate a succession of zero or more instances of the enclosed. Because each instance of the syntactic category $\langle \text{adop} \rangle$ is a single character, we derive

$$\langle \text{boexp} \rangle ::= \langle \text{adder} \rangle \langle \text{boterm} \rangle \quad (15)$$

from which follows $(\text{adder}(L) \text{ and } \text{boterm}(K)) \Rightarrow \text{boexp}(LK)$ (16)

But this gives us no way of proving that a string is not of the syntactic category $\langle \text{boexp} \rangle$. In particular, the conclusion

$$(\text{adder}(L) \text{ and } \text{non } \text{boterm}(K)) \Rightarrow \text{non } \text{boexp}(LK) \quad \text{is not justified.}$$

We must make --in analogy to (6)-- an assumption about $\langle \text{term} \rangle$ and $\langle \text{adop} \rangle$,

and we assume $(\text{term}(L) \text{ and } \text{adop}(y)) \Rightarrow \text{non } \text{boterm}(Ly)$ (17)

This means, to start with, that with $\text{term}(L)$, $\text{term}(L')$, $\text{adop}(y)$, and $\text{adop}(y')$, we can conclude from $LyS = L'y'S'$, that $L = L'$ and $y = y'$. In other words, for every $\langle \text{boexp} \rangle$ that starts with an instance of $\langle \text{term} \rangle \langle \text{adop} \rangle$, that instance is uniquely defined. By removing it from the front end, we are still left with a string from the syntactic category $\langle \text{boexp} \rangle$, and therefore we are allowed to conclude

$$(\text{adder}(L) \text{ and } \text{non } \text{boexp}(K)) \Rightarrow \text{non } \text{boexp}(LK) \quad (18)$$

This does not solve our problems yet, because, in order to use (18) in order to prove $\text{non } \text{boexp}(LK)$, we still have to prove $\text{non } \text{boexp}(K)$, be it only for a possibly shorter string K . We can do it, however, for a string related to the syntactic category $\langle \text{term} \rangle$, as we can prove

$$(\text{boterm}(L) \text{ and } \text{non } \text{boterm}(Ly) \text{ and } \text{boexp}(Ly)) \Rightarrow (\text{term}(L) \text{ and } \text{adop}(y)) \quad (19)$$

The nonempty string Ly , satisfying $\text{boexp}(Ly)$ can have one of three different forms:

1) $\langle \text{term} \rangle \langle \text{adop} \rangle \langle \text{nonempty boexp} \rangle$

This would imply, that L itself is of the form

$$\langle \text{term} \rangle \langle \text{adop} \rangle \langle \text{boexp} \rangle$$

which, on account of its first two elements and (17) is incompatible with

boterm(L)

2) < term > < adop >

Because all instances of < adop > are single characters, this case implies indeed $\text{term}(L) \text{ and } \text{adop}(y)$

3) < boterm >

This case is incompatible with $\text{non boterm}(Ly)$.

Hence, formula (19) has been proved.

Similarly, we should ask ourselves how to prove that some string is not an element of the syntactic category < exp >. From (12) we can derive

$$(\text{adder}(L) \text{ and } \text{term}(K)) \Rightarrow \text{exp}(LK) \quad (20)$$

but, again, the conclusion

$$(\text{adder}(L) \text{ and } \text{non term}(K)) \Rightarrow \text{non exp}(LK) \quad \text{is not justified,}$$

only --similar to (18)--

$$(\text{adder}(L) \text{ and } \text{non exp}(K)) \Rightarrow \text{non exp}(LK) \quad (21)$$

Analogous to (19) we have

$$(\text{boterm}(L) \text{ and } \text{exp}(L)) \Rightarrow \text{term}(L) \quad (22)$$

The term $\text{exp}(L)$ tells us that the string L can have one of two different forms:

1) < term >

This case indeed implies $\text{term}(L)$

2) < nonempty adder > < term >

On account of (17) --and also (4)-- this case is excluded by $\text{boterm}(L)$.

Hence formula (22) has been proved.

Finally we can conclude that

$$(\text{exp}(L) \text{ and } \text{adop}(y)) \Rightarrow \text{adder}(Ly) \quad (23)$$

The left-hand side tells us on account of (12) that Ly is of the form

$$\langle \text{adder} \rangle \langle \text{term} \rangle \langle \text{adop} \rangle$$

and therefore (13) allows us to conclude $\text{adder}(Ly)$, and (23) has been proved.

Syntax rules (12) and (13) strongly suggest that the body of expsearch should call --possibly repeatedly-- a new primitive termsearch . In order to design expsearch in terms of termsearch we only need to know the net effect

of termsearch and we propose --in analogy to (7) and (11)-- that, when T is defined as the string of characters moved over by termsearch, the primitive termsearch will establish $Rt(T, x, c)$, where $Rt(T, x, c)$ is given by

$$\text{boterm}(T) \text{ and } \text{non boterm}(Tx) \text{ and } c = \text{term}(T) \quad (24)$$

Designing expsearch in terms of termsearch means that we would like to have theorems allowing us to draw conclusions from the truth of a relation of the form Rt .

Theorem 4. $(\text{adder}(L) \text{ and } Rt(T, x, c) \text{ and } c \text{ and } \text{adop}(x)) \Rightarrow \text{adder}(LTx)$

Proof. Assumed:

0. $\text{adder}(L) \text{ and } Rt(T, x, c) \text{ and } c \text{ and } \text{adop}(x)$

Derived:

1. $c = \text{term}(T)$ with (24) from 0
 2. c from 0
 3. $\text{term}(T)$ from 1 and 2
 4. $\text{adder}(L)$ from 0
 5. $\text{exp}(LT)$ with (20) from 3 and 4
 6. $\text{adop}(x)$ from 0
 7. $\text{adder}(LTx)$ with (23) from 5 and 6
- (End of Proof of Theorem 4.)

Theorem 5. $(\text{adder}(L) \text{ and } Rt(T, x, c) \text{ and } \text{non } c) \Rightarrow \text{Re}(LT, x, c)$

Proof. Assumed:

0. $\text{adder}(L) \text{ and } Rt(T, x, c) \text{ and } \text{non } c$

Derived:

1. $c = \text{term}(T)$ with (24) from 0
2. $\text{non } c$ from 0
3. $\text{non } \text{term}(T)$ from 1 and 2
4. $\text{boterm}(T)$ with (24) from 0
5. $\text{non } \text{boterm}(Tx)$ with (24) from 0
6. $\text{non } \text{boexp}(Tx)$ with (19) from 3, 4, and 5
7. $\text{adder}(L)$ from 0
8. $\text{non } \text{boexp}(LTx)$ with (18) from 6 and 7
9. $\text{boexp}(LT)$ with (16) from 4 and 7
10. $\text{non } \text{exp}(T)$ with (22) from 3 and 4
11. $\text{non } \text{exp}(LT)$ with (21) from 7 and 10
12. $c = \text{exp}(LT)$ from 2 and 11

13. $Re(LT, x, c)$ with (11) from 8, 9, and 12
(End of Proof of Theorem 5.)

Theorem 6. ($adder(L)$ and $Rt(T, x, c)$ and non $adop(x)$) $\Rightarrow Re(LT, x, c)$

Proof. Assumed:

0. $adder(L)$ and $Rt(T, x, c)$ and non $adop(x)$

Derived:

1. $boterm(T)$ with (24) from 0
2. $adder(L)$ from 0
3. $boexp(LT)$ with (16) from 1 and 2
4. non $boterm(Tx)$ with (24) from 0
5. non $adop(x)$ from 0
6. non $boexp(Tx)$ with (19) from 1, 4, and 5
7. non $boexp(LTx)$ with (18) from 2 and 6
8. $c = term(T)$ with (24) from 0
9. $c \Rightarrow term(T)$ from 8
10. $c \Rightarrow exp(LT)$ with (20) from 2 and 9
11. non $c \Rightarrow$ non $term(T)$ from 8
12. non $c \Rightarrow$ non $exp(T)$ with (22) from 1 and 11
13. non $c \Rightarrow$ non $exp(LT)$ with (21) from 2 and 12
14. $c = exp(LT)$ from 10 and 13
15. $Re(LT, x, c)$ with (11) from 3, 7, and 14
(End of Proof of Theorem 6.)

A corollary of Theorems 5 and 6 is

$$(adder(L) \text{ and } Rt(T, x, c) \text{ and non } (c \text{ and } adop(x))) \Rightarrow Re(LT, x, c)$$

A possible body for `expsearch` is by now pretty obvious when we realize that its calls on `termsearch` imply for its ghost variable `E` the assignment $E := ET$ (as "move" implies $E := Ex$). In the post-assertions for calls on `termsearch` the relation $E = LT$ has been given in order to define `L` in terms of `E` and `T`.

```
proc expsearch: {adder(E) because E = empty string}
                termsearch {E = LT and adder(L) and Rt(T, x, c)};
                do c and adop(x)  $\rightarrow$  {adder(Ex)}
                    move {adder(E)};
                    termsearch {E = LT and adder(L) and Rt(T, x, c)}
                od {Re(E, x, c)}
```

corp

We now consider for $\langle \text{term} \rangle$ the following syntax

$$\langle \text{term} \rangle ::= \langle \text{plier} \rangle \langle \text{prim} \rangle \quad (25)$$

$$\langle \text{plier} \rangle ::= \{ \langle \text{prim} \rangle \langle \text{mult} \rangle \} \quad (26)$$

$$\langle \text{mult} \rangle ::= * \quad (27)$$

and assume about $\langle \text{prim} \rangle$ and $\langle \text{mult} \rangle$

$$(\text{prim}(L) \text{ and } \text{mult}(y)) \Rightarrow \text{non boprim}(Ly) \quad (28)$$

Formulae (25), (26), (27), and (28) are similar to (12), (13), (14), and (17) respectively, and all our conclusions since then carry over. With P as the string of characters moved over by a primitive `primsearch` that establishes --in analogy to (24)-- $R_p(P, x, c)$, where $R_p(P, x, c)$ is given by

$$\text{boprim}(P) \text{ and non boprim}(Px) \text{ and } c = \text{prim}(P) \quad (29)$$

we can write immediately (!)

```
proc termsearch: {plier(T) because T = empty string}
  primsearch {T = LP and plier(L) and Rp(P, x, c)};
  do c and mult(x) → {plier(Tx)}
    move {plier(T)};
    primsearch {T = LP and plier(L) and Rp(P, x, c)}
  od {Rt(T, x, c)}
```

corp

It is time to "close" our syntax:

$$\langle \text{prim} \rangle ::= \langle \text{idem} \rangle \mid \langle \text{paren} \rangle \quad (30)$$

$$\langle \text{idem} \rangle ::= \{ \langle \text{letter} \rangle \} \langle \text{letter} \rangle \quad (31)$$

$$\langle \text{paren} \rangle ::= \langle \text{open} \rangle \langle \text{exp} \rangle \langle \text{close} \rangle \quad (32)$$

$$\langle \text{open} \rangle ::= (\quad (33)$$

$$\langle \text{close} \rangle ::=) \quad (34)$$

$$\langle \text{letter} \rangle ::= a \mid b \mid c \mid d \mid e \mid f \quad (35)$$

The important conclusions from (35) are:

- 1) that the syntactic category $\langle \text{letter} \rangle$ is nonempty
- 2) that all instances of the syntactic category $\langle \text{letter} \rangle$ are all single characters
- 3) that these characters differ from the six previously introduced

characters.

From the nonemptiness of the syntactic category $\langle \text{letter} \rangle$ we draw the same conclusion for $\langle \text{iden} \rangle$, hence for $\langle \text{prim} \rangle$, hence for $\langle \text{term} \rangle$ hence for $\langle \text{exp} \rangle$, and hence for $\langle \text{sent} \rangle$. In particular we shall need to refer to

$$\text{boprim}(\text{empty string}) \quad (36)$$

From (30) we derive

$$\langle \text{boprim} \rangle ::= \langle \text{boiden} \rangle \mid \langle \text{boparen} \rangle \quad (37)$$

From (31) and (32) respectively, we derive

$$(\text{boiden}(y) = \text{letter}(y)) \quad \underline{\text{and}} \quad \underline{\text{non}} \quad \text{iden}(\text{empty string}) \quad (38)$$

$$(\text{boparen}(y) = \text{open}(y)) \quad \underline{\text{and}} \quad \underline{\text{non}} \quad \text{paren}(\text{empty string}) \quad (39)$$

and hence

$$\text{boprim}(y) = (\text{letter}(y) \quad \underline{\text{or}} \quad \text{open}(y)) \quad (40)$$

$$\underline{\text{non}} \quad \text{prim}(\text{empty string}) \quad (41)$$

From (31) we derive

$$\langle \text{boiden} \rangle ::= \{ \langle \text{letter} \rangle \} \quad (42)$$

and, because instances of $\langle \text{letter} \rangle$ are single characters

$$\underline{\text{non}} \quad \text{letter}(y) \Rightarrow \underline{\text{non}} \quad \text{boiden}(Ly) \quad (43)$$

From (32) we derive

$$\langle \text{boparen} \rangle ::= \text{empty string} \mid \langle \text{open} \rangle \langle \text{boexp} \rangle \mid \langle \text{paren} \rangle \quad (44)$$

The three alternatives for $\langle \text{boparen} \rangle$ are mutually exclusive: for the first one versus the two others, it is obvious. For the last two I can prove the mutual exclusion only by using the technique of the bracket count.

Lemma 1. $\text{exp}(L)$ implies that the number of instances of $\langle \text{open} \rangle$ in L equals the number of instances of $\langle \text{close} \rangle$ in L .

Lemma 2. $\text{boexp}(L)$ implies that the number of instances of $\langle \text{open} \rangle$ in L equals at least the number of instances of $\langle \text{close} \rangle$ in L .

Lemma 1 follows from the fact that in the original syntax --i.e. without the "begin-of"-derivations-- the only rule using $\langle \text{open} \rangle$ or $\langle \text{close} \rangle$, viz. (32), introduces them pairwise. Lemma 2 follows from the observation that in this only introduction, the instance of $\langle \text{open} \rangle$ precedes that of $\langle \text{close} \rangle$

(Presumably official syntactic theory has more formal proofs for these two Lemmata; I am fully convinced of their correctness by the preceding four lines of argument.)

The last two alternatives of (44) are mutually exclusive, because from Lemma 2 we can conclude that in a string of the form $\langle \text{open} \rangle \langle \text{boexp} \rangle$ the number of instances of $\langle \text{open} \rangle$ exceeds the number of instances of $\langle \text{close} \rangle$, while in a string of the form $\langle \text{paren} \rangle$ these numbers are equal on account of Lemma 1. In other words:

$$(\text{open}(y) \text{ and } \text{boexp}(L)) \Rightarrow \text{non } \text{paren}(yL) \quad (45)$$

or, equivalently

$$\text{paren}(yL) \Rightarrow (\text{open}(y) \text{ and } \text{non } \text{boexp}(L)) \quad (45')$$

Expressed in terms of paren and boparen only, also holds

$$\text{paren}(L) \Rightarrow \text{non}(\exists z: \text{boparen}(Lz)) \quad (46)$$

This formula can be derived by deriving a contradiction from the truth of the left-hand side and the falsity of the right-hand side. From $\text{paren}(L)$ and (39) we conclude that L is nonempty, and we may write $L = yK$, such that, on account of (45'), we deduce

$$\text{open}(y) \text{ and } \text{non } \text{boexp}(K)$$

On the other hand, $(\exists z: \text{boparen}(yKz))$ is, according to (1), equivalent to

$$(\exists z, M: \text{paren}(yKzM))$$

or

$$(\exists M, z: \text{paren}(yKMz))$$

Rule (32) then allows us to conclude

$$\text{open}(y) \text{ and } (\exists M: \text{exp}(KM)) \text{ and } (\exists z: \text{close}(z)) \quad .$$

The second term is equivalent to $\text{boexp}(K)$, we have the contradiction we were looking for, and hence, (46) has been proved.

Theorem 7. $(L = \text{empty string and non}(\text{letter}(x) \text{ or } \text{open}(x))) \Rightarrow \text{Rp}(L, x, \text{false})$

Proof. Assumed:

0. $L = \text{empty string and non}(\text{letter}(x) \text{ or } \text{open}(x))$

Derived:

1. $L = \text{empty string}$ from 0
2. $\text{boprim}(L)$ with (36) from 1
3. $\text{non}(\text{letter}(x) \text{ or } \text{open}(x))$ from 0

4. non boprim(x) with (40) from 3
 5. $x = Lx$ from 1
 6. non boprim(Lx) from 4 and 5
 7. $false = prim(L)$ with (41) from 1
 8. $Rp(L, x, false)$ with (29) from 2, 6, and 7
- (End of Proof of Theorem 7.)

Theorem 8. (iden(yL) and letter(x)) \Rightarrow iden(yLx)

Proof. Evident from (31)

Theorem 9. (iden(yL) and non letter(x)) \Rightarrow Rp(yL, x, true)

Proof. Assumed:

0. iden(yL) and non letter(x)
Derived:
 1. iden(yL) from 0
 2. boiden(yL) with (5) from 1
 3. boprim(yL) with (37) from 2
 4. boiden(y) with (4) from 2
 5. letter(y) with (38) from 4
 6. non open(y) from 5
 7. non boparen(y) with (39) from 6
 8. non boparen(yLx) with (4) from 7
 9. non letter(x) from 0
 10. non boiden(yLx) with (43) from 9
 11. non boprim(yLx) with (37) from 8 and 10
 12. true = prim(yL) with (30) from 1
 13. Rp(yL, x, true) with (29) from 3, 11, and 12
- (End of Proof of Theorem 9.)

See Note 4 on page
EWD550 - 15

Theorem 10. (open(y) and Re(E, x, c) and c and close(x)) \Rightarrow Rp(yEx, z, c)

Proof. Assumed:

0. open(y) and Re(E, x, c) and c and close(x)
Derived:
1. $c = exp(E)$ with (11) from 0
2. c from 0
3. $exp(E)$ from 1 and 2
4. open(y) from 0
5. close(x) from 0
6. paren(yEx) with (32) from 3, 4, 5

- | | | |
|-----|-----------------------------------|------------------------------|
| 7. | $\text{prim}(yEx)$ | with (30) from 6 |
| 8. | $\text{boprim}(yEx)$ | with (5) from 7 |
| 9. | <u>non</u> $\text{boparen}(yExz)$ | with (46) from 6 |
| 10. | <u>non</u> $\text{letter}(y)$ | from 4 |
| 11. | <u>non</u> $\text{boiden}(y)$ | with (38) from 10 |
| 12. | <u>non</u> $\text{boiden}(yExz)$ | with (4) from 11 |
| 13. | <u>non</u> $\text{boprim}(yExz)$ | with (37) from 9 and 12 |
| 14. | $c = \text{prim}(yEx)$ | from 2 and 7 |
| 15. | $\text{Rp}(yEx, z, c)$ | with (29) from 8, 13, and 14 |
- (End of Proof of Theorem 10.)

Theorem 11. (open(y) and $\text{Re}(E, x, c)$ and non c) \Rightarrow $\text{Rp}(yE, x, c)$

Proof. Assumed:

0. open(y) and $\text{Re}(E, x, c)$ and non c

Derived:

- | | | |
|-----|----------------------------------|-----------------------------|
| 1. | $\text{boexp}(E)$ | with (11) from 0 |
| 2. | <u>open</u> (y) | from 0 |
| 3. | $\text{boparen}(yE)$ | with (44) from 1 and 2 |
| 4. | $\text{boprim}(yE)$ | with (37) from 3 |
| 5. | <u>non</u> $\text{letter}(y)$ | from 2 |
| 6. | <u>non</u> $\text{boiden}(y)$ | with (38) from 5 |
| 7. | <u>non</u> $\text{boiden}(yEx)$ | with (4) from 6 |
| 8. | <u>non</u> $\text{boexp}(Ex)$ | with (11) from 0 |
| 9. | $c = \text{exp}(E)$ | with (11) from 0 |
| 10. | <u>non</u> c | from 0 |
| 11. | <u>non</u> $\text{exp}(E)$ | from 9 and 10 |
| 12. | <u>non</u> $\text{paren}(yEx)$ | with (32) from (2 and) 11 |
| 13. | <u>non</u> $\text{boparen}(yEx)$ | with (44) from 8 and 12 |
| 14. | <u>non</u> $\text{boprim}(yEx)$ | with (37) from 7 and 13 |
| 15. | <u>non</u> $\text{boiden}(yE)$ | with (4) from 6 |
| 16. | <u>non</u> $\text{iden}(yE)$ | with (5) from 15 |
| 17. | <u>non</u> $\text{paren}(yE)$ | with (45) from 1 and 2 |
| 18. | <u>non</u> $\text{prim}(yE)$ | with (30) from 16 and 17 |
| 19. | $c = \text{prim}(yE)$ | from 10 and 18 |
| 20. | $\text{Rp}(yE, x, c)$ | with (29) from 4, 14 and 19 |
- (End of Proof of Theorem 11.)

Theorem 12. $(\text{open}(y) \text{ and } \text{Re}(E, x, c) \text{ and } \text{non close}(x)) \Rightarrow \text{Rp}(yE, x, \text{false})$

Proof. Assumed:

0. $\text{open}(y) \text{ and } \text{Re}(E, x, c) \text{ and } \text{non close}(x)$
 Derived:
1. $\text{boexp}(E)$ with (11) from 0
 2. $\text{open}(y)$ from 0
 3. $\text{boparen}(yE)$ with (44) from 1 and 2
 4. $\text{boprim}(yE)$ with (37) from 3
 5. $\text{non letter}(y)$ from 2
 6. $\text{non boiden}(y)$ with (38) from 5
 7. $\text{non boiden}(yEx)$ with (4) from 6
 8. $\text{non boexp}(Ex)$ with (11) from 0
 9. $\text{non close}(x)$ from 0
 10. $\text{non paren}(yEx)$ with (32) from 9
 11. $\text{non boparen}(yEx)$ with (44) from 8 and 10
 12. $\text{non boprim}(yEx)$ with (37) from 7 and 11
 13. $\text{non boiden}(yE)$ with (4) from 6
 14. $\text{non iden}(yE)$ with (5) from 13
 15. $\text{non paren}(yE)$ with (45) from 1 and 2
 16. $\text{false} = \text{prim}(yE)$ with (30) from 14 and 15
 17. $\text{Rp}(yE, x, \text{false})$ with (29) from 4, 12, and 16

Note 4. In proofs 9 through 12, I refer a number of times to formula (4), but it is not really that one that is needed, but the obvious generalization

$$\text{bopqr}(KL) \Rightarrow \text{bopqr}(K) ;$$

sometimes it is used in the inverted, but equivalent, form

$$\text{non bopqr}(K) \Rightarrow \text{non bopqr}(KL) .$$

Furthermore I offer my apologies for the great similarity between the proofs of Theorem 11 and Theorem 12. The total text could have been shortened by first stating a Lemma 3 that captures the intersection of the two proofs. It is just too expensive to change in this respect this document, which is not intended to be submitted for publication. (End of Note 4.)

With Theorems 7 through 12 we have prepared the way for the following design of a body for `primsearch` .


```

proc primsearch : {P = empty string}
  if non(letter(x) or open(x)) → {Rp(P, x, false)}
    c := false {Rp(P, x, c)}
    [] letter(x) → move {P = yL and iden(P)};
    do letter(x) → {P = yL and iden(Px)}
      move {P = yL and iden(P)}
    od {Rp(P, x, true)};
    c := true {Rp(P, x, c)}
  [] open(x) → move {P = y and open(y)};
  expsearch {P = yE and open(y) and Re(E, x, c)};
  if c and close(x) → {Rp(Px, z, c)}
    move {Rp(P, x, c)}
    [] non c → skip {Rp(P, x, c)}
    [] non close(x) → {Rp(P, x, false)}
      c := false {Rp(P, x, c)}
  fi {Rp(P, x, c)}
fi {Rp(P, x, c)}

```

corp

Now our syntax has been "closed" by (30) through (35), we can at last fulfill our obligation of proving what up till now have been assumptions, viz.

$$\text{sent}(L) \Rightarrow \text{non } (\exists y: \text{bosent}(Ly)) \quad (6)$$

$$(\text{term}(L) \text{ and adop}(y)) \Rightarrow \text{non } \text{boterm}(Ly) \quad (17)$$

$$(\text{prim}(L) \text{ and mult}(y)) \Rightarrow \text{non } \text{boprim}(Ly) \quad (28)$$

Relation (6) follows from the fact that $\text{bosent}(Ly)$ implies $\text{boexp}(L)$, and from our syntax for $\langle \text{exp} \rangle$, $\langle \text{term} \rangle$, $\langle \text{prim} \rangle$, and $\langle \text{iden} \rangle$, this implies that L does not contain a semicolon; $\text{sent}(L)$ implies according to (8) that L does contain a semicolon. This is the contradiction that follows from the assumption that (6) does not hold; hence (6) has been proved. In order to prove (17) --under the assumption of (28)!-- we observe that with

$$\langle \text{term} \rangle ::= \langle \text{plier} \rangle \langle \text{prim} \rangle$$

$$\langle \text{boterm} \rangle ::= \langle \text{plier} \rangle \langle \text{boprim} \rangle$$

the negation of (17)

$$\text{term}(L) \text{ and adop}(y) \text{ and boterm}(Ly)$$

would imply that $\langle \text{prim} \rangle \langle \text{adop} \rangle$ could be of the form $\langle \text{boprim} \rangle$. It

therefore suffices to prove that

$$(\text{prim}(L) \text{ and } \text{op}(y)) \Rightarrow \text{non boprim}(Ly) \quad \text{with} \\ \langle \text{op} \rangle ::= \langle \text{adop} \rangle \mid \langle \text{mult} \rangle .$$

This last implication can be proved by deriving a contradiction from its negation:

$$\text{prim}(L) \text{ and } \text{op}(y) \text{ and } \text{boprim}(Ly) \quad ;$$

it can be done using Lemma 1 and Lemma 2, and I gladly leave this detail to the reader.

* * *

In view of the length of this report, the transformation from infix to postfix notation --on page EWD550 - 0 announced as "our ultimate goal"!-- will be postponed and left for some later document.

History.

Nearly three years ago I wrote a seven-page report, EWD375 "A non algebraic example of a constructive correctness proof." in which (essentially) the same problem has been tackled as here. Last January, while I was lecturing in La Jolla, Jack Mazola urged me to show a more complicated example; I tried to reconstruct the argument of EWD375 on the spot and failed.

Last February, when I was home again, I reread EWD375 and it left me greatly dissatisfied. I remembered that EWD375 had been a cause for great enthusiasm when it was written, and I could not understand that enthusiasm anymore. I found EWD375 very hard to read and hardly convincing: what three years ago I had considered as "a proof" now struck me at best as "helpful heuristics". (A strange experience to be nearly ashamed of what had been a source of pride only a few years ago!)

It was now clear why, last January in La Jolla, I was unable to give on the spot a formal treatment of the syntax analyzer: it was not just a failure of memory, it was also a profound change in my standards of rigor (undoubtedly also caused by the fact that over the last few years I burned a few fingers!). I decided to forget EWD375 and to start again from scratch. This document is the result of that effort.

It has been surprisingly hard to write. After the first six pages had been written --I had only dealt with sentsearch-- there has been a long pause before I gathered the strength and the courage to tackle expsearch, and for a few weeks I put the unfinished document away. To undertake the treatment of primsearch proved to be another hurdle.

What the final document does not show is that the notation eventually used for the assertions, the theorems and the proofs is the result of many experiments. Before we invented, for instance, the trick to use the predicate $pqr(K)$ to denote that the string K belongs to the syntactic category $\langle pqr \rangle$ all our formulae became unwieldy; so they did, as long as we indicated concatenation of strings with an explicit operator instead of --as eventually-- just by juxtaposition. I hesitated, when I wrote --as on the middle of page EWD550 - 3-- $\text{sent}(L;)$ because I saw problems coming by the time that I had to write such predicates for strings containing unmatched parentheses; the trick of introducing $\langle \text{open} \rangle$ and $\langle \text{close} \rangle$ solved that problem. Instead of (8) I should have written

$$\begin{aligned} \langle \text{sent} \rangle &::= \langle \text{exp} \rangle \langle \text{semi} \rangle \\ \langle \text{semi} \rangle &::= ; \end{aligned}$$

Again, at the time of writing, also this report has been a source of great excitement. This is somewhat amazing as it does not contain a single deep thought! Is it, because we now still remember how much more beautiful it is than all the rejected efforts? I wonder how I shall feel about it in a few years time!

Acknowledgement.

I am greatly indebted to W.H.J. Feijen, M. Rem, A.J. Martin and C.S. Scholten, whose encouragement and active participation have been absolutely essential. And I am grateful to Jack Mazola for providing me with the incentive.

19th of March 1976
Burroughs
Plataanstraat 5
NUENEN - 4565
The Netherlands

prof.dr. Edsger W. Dijkstra
Burroughs Research Fellow