On iteration and recursion

It is customary – I know, in fact, of no other way – to define iteration recursively. But the technique, though as sound as sound can be, has a property that might be very nasty, at least from a methodological point of view. (And if it turns out not to be that nasty, it is at least annoying.)

I can already illustrate what I have in mind with the syntax for natural numbers, for which there are at least three forms:

<natural number> ::= <digit> | <natural number> <digit>

<natural number> ::= <digit> | <digit> <natural number>

<natural number> ::=
    <digit> | <natural number> < natural number> .

In a way the last syntax is perhaps the most honest. It leads, however, to an ambiguous parsing of all natural numbers of at least 3 digits: it defines parsing as a nondeterministic process. The first two syntaxes represent the two usual ways of resolving that nondeterminacy. The purpose of this note is to explore how much that choice can hurt.

It is quite possible that most of the exploration can, indeed, be carried out in terms of the above example. But, firstly, I am not an expert in automatic compiler generation, and, secondly, my interest in the question arose in connection with other associative operations than concatenation.

We encounter the same phenomenon with iterated functional composition, with

$$f^0 x = x \quad , \text{ supplemented with}$$

$$f^{n+1} x = f\,(f^n x) \quad \text{or with}$$

$$f^{n+1} x = f^n (f\,x) \quad \text{or with}$$

$$f^{n+m} x = f^n (f^m x) \quad \text{and} \quad f^1 x = f\,x \quad .$$

We also encounter it in programming languages like SASL, in which we can define -with my syntax-

```
chip n (a:b) = if n=0 → a:b
               [] n>0 → chip (n-1) b
               fi ,

chop n x = if n=0 → x
           [] n>0 → q where
                    p:q = chop (n-1) x
           fi .
```

(For the sake of comparison it would perhaps have been nicer to define chip by the trivially equivalent text

```
chip n x = if n=0 → x
           [] n>0 → chip (n-1) b where
                    a:b = x
           fi .)
```

Clearly chip and chop are the same functions, and it does not seem unreasonable to require from implementa-

tions that they evaluate chip and chop at the same expense (in terms of time, space, and amount of garbage generated). In this note I intend to explore whether theorems about the one are more easily proved than about the other. (I hope there is no difference, but if there is, we had better know.)

In terms of tl - for "tail" - they are defined by

```
chip 0 x = x
chip (n+1) x = chip n (tl x)
```

```
chop 0 x = x
chop (n+1) x = tl (chop n x)
```

Lemma 0A

chip n (tl x) = tl (chip n x)

Proof By induction; n=0 is obvious.

```
  chip (n+1) (tl x)
= {definition of chip}
  chip n (tl (tl x))
= {induction hypothesis}
  tl (chip n (tl x))
= {definition of chip}
  tl (chip (n+1) x)
```

QED

Lemma 0B

chop n (tl x) = tl (chop n x)

Proof By induction; n=0 is obvious.

```
  chop (n+1) (tl x)
= {definition of chop}
  tl (chop n (tl x))
= {induction hypothesis}
  tl (tl (chop n x))
= {definition of chop}
  tl (chop (n+1) x)
```

QED.

I have carried out those two silly proofs side by side for the sake of comparison: they are equally simple.

The awkward thing is that I could <u>not</u> find a proof for chip n x = chop n x that was symmetric in chip and chop! I needed either Lemma 0A or Lemma 0B. I was very much annoyed when I found myself forced to treat an obviously symmetric situation asymmetrically.

Let us now look at the proofs of

chip n (twice x) = twice (chip n x) and
chop n (twice x) = twice (chop n x)

with twice (a:b) = 2*a : twice b , from which we immediately derive tl (twice x) = twice (tl x) . The theorems hold for n=0 ; we only need to compare the induction steps.

```
  chip (n+1) (twice x)
= {definition of chip}
  chip n (tl (twice x))
= {property of twice}
  chip n (twice (tl x))
= {induction hypothesis}
  twice (chip n (tl x))
= {definition of chip}
  twice (chip (n+1) x)
                    QED.
```

```
  chop (n+1) (twice x)
= {definition of chop}
  tl (chop n (twice x))
= {induction hypothesis}
  tl (twice (chop n x))
= {property of twice}
  twice (tl (chop n x))
= {definition of chop}
  twice (chop (n+1) x)
                    QED.
```

Again, the two proofs are equally simple. From the above experience I am willing to extrapolate that proofs in terms of chip are as straightforward as those in terms of chop. But now it becomes profoundly annoying that they are different. The cleanest way out is undoubtedly to establish for iterated function application <u>immediately</u>

$$f\ (m+n)\ x\ =\ f\ m\ (f\ n\ x)$$

and to use <u>that</u> equality in <u>all</u> subsequent proofs. Thus all subsequent arguments about iteration – and we must encounter many of them – are insensitive to the irrelevant

choice whether iteration has been originally defined in the style of chip or in the style of chop. It would then be honest to define

```
choip n (a:b) =
    if n=0 → a:b
    [] n=1 → b
    [] p,q: p>0 ∧ q>0 ∧ p+q=n →
            choip p (choip q (a:b))
    fi
```

where I have taken the liberty of using an "initializing guard" in the last guarded expression.

Plataanstraat 5
5671 AL NUENEN
The Netherlands

14 April 1982
prof. dr. Edsger W. Dykstra
Burroughs Research Fellow.