On the productivity of recursive definitions.

The purpose of this note is to summarize what we have learned at the last two sessions of the Tuesday Afternoon Club. We shall discuss conditions under which finite expressions can be said to represent infinite sequences. The simplest example of such a finite expression is

ones

where ones is defined recursively by

<u>def</u> ones = 1 : ones .

(Here we have borrowed from SASL the colon to denote concatenation.)

Repeated application of the definition of ones gives

```
  ones
= 1 : ones
= 1 : 1 : ones    , and so on.
```

On terminology.

All through our discussions we have been painfully aware of our need of a terminology that would not be misleading.

<u>Aside</u>. How much harm can be done by a misleading terminology was brought home to me this morning, when I received a technical note (without title) written by Richard J. Treitel, a Stanford student.

When I introduced the predicate transformer, I knew that it wasn't fully correct to call it "wp", and that it wasn't fully correct either to denote its (second) argument and its value by "post-condition" and "pre-condition" respectively. In view of the convention $\{P\}\ S\ \{R\}$, I could have called them "right-condition" and "left-condition". Yet I adopted the traditional terminology, hoping that no-one would be misled. But R.J. Treitel states about wp that "it explicitly involves a notion of time". (End of Aside.)

For our purposes, the term "infinite" won't do. Firstly, when introducing two complementary concepts, I prefer "positive" terms for both, because naming the one by the negation of the other destroys the symmetry by presenting the latter one as the more "basic" concept. (For $x \neq y$ I prefer the pronunciation "x differs from y" to "x is not equal to y". I have still to encounter the first mathematician who pronounces $x = y$ as "x doesn't differ from y"!) Secondly, we shall encounter two drastically different ways in which a sequence can fail to be infinite.

In the time domain we have —remarkably enough!— plenty of negation-free terms to denote infinite, such as "eternal" and "permanent", but that won't help us, for my ultimate goal is a non-operational treatment in which our texts needn't be interpreted as executable code. I have hesitated for a long time between "ongoing sequences" and "continuing sequences" —not being too pleased with either of the two—. After consultation of all my English and American dictionaries I concluded that "continued concatenation" should do the job.

Note. My Webster's New Collegiate Dictionary gives: "continued fraction" *n*: a fraction whose numerator is an integer and whose denominator is an integer plus a fraction whose numerator is an integer and whose denominator is an integer plus a fraction and so on".

The Shorter Oxford English Dictionary gives "Continued 1 [.....] 2. Carried on in space, time, or series" with as one of the examples: "C. fraction: a fraction whose denominator is an integer plus a fraction, which latter fraction has a similar denominator, and so on."

(In passing I noted that none of my dictionaries defined "continued fraction: a fraction whose numerator is an integer and whose denominator is an integer plus a continued fraction.")

I take the fact that dictionaries from both sides of the Atlantic Ocean close their definitions with the famous "and so on" as an indication that "continued" admirably renders the idea of a non-terminating recursion.

(End of "On terminology".)

* * *

In the following, <atom> stands for a character from a (finite or infinite) alphabet, say a natural number. Sequences, concatenations, and continued concatenations are defined by the following syntax:

```
<seq> ::= <atom> | <conc>
<conc> ::= (<seq> : ⪡ seq ⪢)
<contconc> ::= (<seq> : ⪡ contconc ⪢)
```

With the bracket pair ⪡ ⪢ I denote an instance of the corresponding syntactic category, but with its outer parenthesis pair, if present, optionally stripped off. (This seemed a convenient way to indicate that concatenation as denoted by the colon is deemed to associate to the right.) Note that I did not bother to introduce the empty sequence (nor NIL).

The usual functions "head" and "tail" are only introduced for concatenations (continued or not):

$$\begin{aligned} \mathrm{hd}\ (p:q) &= p \\ \mathrm{tl}\ (p:q) &= q \qquad ; \end{aligned}$$

a head is always a <seq>, and a tail is a <seq> or a <contconc>, just as the argument.

For concatenations we define the true prefix "tpf n" of length n for $n \geq 1$ by

$$\begin{aligned} \mathrm{tpf}\ 1\ (p:q) &= p \\ \mathrm{tpf}\ (n+1)\ (p:q) &= (p:(\mathrm{tpf}\ n\ q)) \qquad ; \end{aligned}$$

a tpf n (p:q) — when defined, of course — is always a <seq>. The distinguishing feature of continued concatenations is that tpf n <contconc> is defined for all $n \geq 1$; tpf n <conc> is only defined for n up to some limit — this, because the true prefix of an <atom> is undefined.

That, for some f, tpf n f is defined for all $n \geq 1$ is, in principle, always proved by mathematical induction over n. With

<u>def</u> ones = 1 : ones

the fact that "tpf n ones" is a <seq> implies that also "tpf (n+1) ones" is a <seq> because "1" is a <seq>. Hence, ones is a continued concatenation.

With <u>def</u> C = c : D ;
<u>def</u> D = d : C ,

one way would be to derive C = c : d : C by substitution and to proceed as before. With "seq f" meaning "f is a <seq>", and with "Q m" meaning "(tpf m C) and (tpf m D) are both defined", it is easily seen that

$$(\text{seq } c) \wedge (\text{seq } d) \wedge (Q\ m) \Rightarrow Q\ (m+1) ,$$

and mathematical induction over the single m will do the job. In the case of mutual recursion this trick seems applicable in general.

More interesting recursive definitions contain a formal parameter, and are essentially of the form

$$\underline{\text{def}}\ f\,x = g\,x : f\,(h\,x) \quad .$$

Given a predicate P such that,

$$(P\,x) \Rightarrow (\text{seq}\ (g\,x)) \wedge (P\,(h\,x)) \qquad (*)$$

(or, with functional application the highest priority:

$$P\,x \Rightarrow \text{seq}\,(g\,x)\ \wedge\ P\,(hx) \qquad)$$

and with "$Q\ m$" meaning "for all x, $P x$ implies that tpf $m\,(f\,x)$ is defined", it is easy to show that $Q\ m \Rightarrow Q\,(m+1)$. Hence formula (*) summarizes our proof obligation for the demonstration that $P x$ implies that $f\,x$ is a continued concatenation.

The generalization to mutually recursive definitions is now straightforward. With

$$\underline{\text{def}}\ f0\ x = g0\ x : f1\,(h1\ x);$$
$$\underline{\text{def}}\ f1\ x = g1\ x : f0\,(h0\ x)$$

the analogue of (*) is

$$P0\ x \Rightarrow \text{seq}\,(g0\ x) \wedge P1\,(h1\ x) \quad \text{and}$$
$$P1\ x \Rightarrow \text{seq}\,(g1\ x) \wedge P0\,(h0\ x) \quad .$$

Finally, some examples.

For
```
def id f = p: id q
           where p:q = f
```
take for P: Px means: x is a continued concatenation

Consider:
```
eleven p = p:p
eleven (p:q)= p: p+p1 : q1
                where p1:q1 = eleven q
```

Here the first line defines eleven for an argument that is an atom, the next two define the value of eleven when the argument is a concatenation. This time the previous P does not suffice; we need now: "Px means: x is a member of the syntactic category <x>, defined by <x> ::= <integer> : {x} ". (For such arguments the first line of the definition is irrelevant.)

In our next example we are interested in a condition Q x such that eleven x is a <seq>. Take for Q: "Q x means that x is a member of the syntactic category <x>, defined by
<x> ::= <integer> | <integer> : {x} ".
We can even prove

$$Q\ x \Rightarrow Q\ (\text{eleven } x)$$

a result we need when we wish to prove that with $\quad$ <u>def</u> pascal x = x : pascal (eleven x)

Q x implies that pascal x is a continued concatenation. (The interesting value is, of course, pascal 1 ; it equals the continued concatenation of the lines of the Pascal Triangle.)

What is the condition on x such that frill x is a continued concatenation, with frill given by

<u>def</u> frill x = p : frill q
$\qquad$ <u>where</u> p:q = eleven x $\qquad$?

<u>Remark</u>. Instead of tpf n , which is not defined on atoms, we should have introduced pf n , given by

pf 1 p = p
pf 1 (p:q) = p
pf (n+1) (p:q) = p: pf n q .

(End of Remark)

Plataanstraat 5 $\qquad$ 18 - 25 September 1980
5671 AL NUENEN $\qquad$ prof. dr. Edsger W. Dijkstra
The Netherlands $\qquad$ Burroughs Research Fellow.