Burroughs

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

Professor C.A.R.Hoare
Department of Computer Science
The Queen's University of Belfast
BELFAST BT7 1NN
Northern Ireland

13 July 1974

Dear Tony,

I feel very guilty for not having given you my reactions to your Stanford report "STAN-CS-73-400: Recursive Data Structures", although I have it already for six moths in my possession. I have read it many times, I liked most of it very much and should have told you so much earlier. Please forgive me my long reaction time.

In a quite different context I had already used very tentatively "equations" as a kind of "guards" that are to be considered true when the equation has a solution, which then can be used in its guarded statement. I had done so successfully, but only in a very restricted environment; to see you doing something very similar in a much more general environment, was a great encouragement for me, for which I am thankful.

Last week I was doing al sorts of little experiments, introducing new data types with a BNF-notation, like

< record file > ::= {< record >} < final record >
< transaction > ::= < deletion > | < update > | < insertion > ;

it seemed to work after a fashion, but as soon as I tried to find the correct generalization of the instance I had constructed, I got in a hopeless mess. So, in utter despair, I picked up again your report "Recursive Data Structures" and immediately saw the trap I had fallen into --in spite of having studied your report a number of times: I am very slow-witted and it takes a lot of time to sink in!--. The trap of BNF, of course, being

Burroughs

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

that all generators, as introduced by you, are kept anonymous. And very quickly I found myself dregged into the adhoccery of introducing all sorts of implied transfer functions and the mess was complete. Yet the stupid exercise taught me something, that I would like to try to explain to you.

I started thinking "why introduce such data type definitions in the first place?". For after all, one only introduces an enumerable set of distinct values, and why not just identify them with the integers 0, 1, 2..? The answer is obvious: because we want to introduce operations on and functions of these values, operations and functions which become utterly chaotic, when expressed in terms of these identifying integers. It is for that purpose that one needs a more adequate terminology for the description of these values. The first moral is, that the introduction of a new data type can only be justified, after the set of operations and functions has been decided upon. (You probably knew this already a long time ago, I even may have read it in your writings; for me it was a good thing to discover this obvious truth in all clarity for myself.)

The next thing I realized is that many such functions --see, for instance, EWD428-- are not mutually independent, and that the separation between "primitive" and "derived" functions, which is then always possible, is often too arbitrary to be attractive. And it is exactly such an arbitrary choice that your data type definitions force upon us, as long as we insist upon each value being generated in a unique way. For instance, with

type string = simple(letter) | conc(string, letter)

the function "last" --last(abcd) = d-- is much better catered for by the syntax than the function "first" --first(abcd) = a--. It does, of course, not help to switch to

type string = simple(letter) | conc(letter, string)

for then we have the misery the other way round. So, I searched for a syntax, that would cater equally well for both functions "first" and "last".

Burroughs

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

The answer is that our syntax must be ambiguous, that our data type definitions must be able to generate the same value in more than one way. This seems in full accordance with well-known types, such as "integer" where we really do not care whether the state x = 21 has been brought about by 17 + 4 or by 13 + 8 : its value is more interesting than the history that created it, the fact that different histories can lead to the same values is exactly the sort of information destruction that seems so characteristic for all meaningful computing.

So I would venture

```
type string = simple(letter) | ass conc(string, string)
```

indicating explicitly that the generator "conc" is associative; as a result I shall admit in my equations "conc" with more than two arguments, when that is convenient. Then

```
last(s:string):letter;
last:= case s of
       (simple(v) → v
       [] conc(u, simple(v)) → v
       )
```

```
first(s:string):letter;
first:= case s of
       (simple(u) → u
       [] conc(simple(u), v) → u
       )
```

and finally

```
sym(s:string):boolean;
sym:= case s of
      (simple(x) → true
      [] conc(simple(x), simple(y)) → x = y
      [] conc(simple(x), y, simple(z)) → x = z and sym(y)
      )
```

With the unique syntax of the previous page, the coding of the body of "sym" becomes a glorious horror, its execution becomes orders of magnitude

Burroughs

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

worse. You should code it, if you have not already done so yourself. (My guess is that the true LISP-addict won't complain, or, still more foolish, will proudly show his optimizing LISP compiler!)

Instead of inserting the reserved characted "ass", telling that the generator "conc" is associative, one could also insert the axiom, say

conc(con(x, y), z) = conc(x, conc(y, z)) ,

but I don't know, what Pandora's Box we then have opened! It then begins to smell like artificial intelligence, a subject, my safe distance from which I have never had reasons to regret.

As you will realize without me pointing that out, we have now non-deterministic functions like

```
rot(s:string):string;
rot:= case s of (conc(u, v) → conc(v, u))
```

(leading to abortion in the case s = simple(x) , for which we have not catered; rot(abc) = bca or cab.) Is it the traditional fear for non-deterministic programming languages that has made LISP the way it is?

The above ideas emerged when I tried to tell Carel S.Scholten --with whom I work now quite regularly for more than 20 years!-- what I liked about your report and in what respect I had my hesitations. The above ideas are therefore perhaps as much his as mine. The possibility of indicating that a generator is a symmetric function has been mentioned, but not explored.

I hope that in terms of "inspiration" I have repaid some of my debt to you!

Yours ever

Edsger

cc.: R.D.Merrell
D.E.Knuth
C.S.Scholten
M.Woodger
(list not necessarily exhaustive)

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow

P.S. I seem to have developed a tendency for skipping my "n"s! END.