Canonical string reduction.

salvo errore
et omissione

This note addresses the problem of how to characterize an intermediate machine state during the evaluation of a SASL expression. Addressing that problem without any specific implementation in mind and observing that both the initial state and the answer are legitimate SASL expressions, I shall try to characterize the relevant intermediate states as SASL expressions as well. In doing so I hope to design such a strict scheme for using scopes that some light is shed on the problem of garbage detection.

I have to adapt SASL's syntax to my purpose. Instead of writing

```
from 0
    where from n = n : from (n+1)
```

I shall write

```
|[ from with from=
     |[ n : from (n+1) with n= ]|
]| 0
```

The inner block on the middle line delineates the scope of n ; the fact that it ends on "with n=]|" indicates that it represents what some readers may know better as the λ-expression $\lambda n.(n: from (n+1))$. The outer block delineates the scope of the identifier from. Its first occurrence states that the whole block stands for the object locally known as "from"; "<u>with</u> from =" introduces a definition of the object locally known as "from"; the object being a function, its definition then follows as a λ-expression. Note the contrast with the original SASL program: there the whole expression "from 0" was defined within the scope of "from", while in our version the scope of "from" is chosen as small as possible.

We shall do our experiments with a slightly more ambitious program, viz. the program for the tabulation of fusc, inspired by J.L.A. van de Snepscheut, designed by A.J.M. van Gasteren, and using the function "zip" as simplified by F.E.J. Kruseman Aretz.

In regular SASL the text could be

```
x
where x = 1 : zip x (elf x)
      where zip (p:q) r = p : zip r q
            elf(a:b:c) = a+b : elf(b:c)
```

The above definition fails to express very directly — i.e. otherwise than by the recognition of the pattern "b:c" — that the tail of the original argument of elf is the argument of the recursive call.

In our notation, we would write

```
|[ x with x = 1 :                                        (0)
   |[zip with zip =                                      (1)
      |[ p :                                             (2)
         |[zip r q with r=]| with p:q =                  (3)
      ]|                                                 (4)
   ]| x                                                  (5)
   ( |[elf with elf =                                    (6)
       |[ a +                                            (7)
           |[ b with b:?=y]| : elf y with a:y =          (8)
       ]|                                                (9)
     ]| x )                                              (10)
]|                                                       (11)
```

The scope of x extends over 0-11; the scope of zip extends over 1-5; the scope of p:q extends over 2-4; the scope of r is on line 3; the scope of elf extends over 6-10; the scope of a:y extends over 7-9; the scope of b is on line 8.

The definition of zip on 2-4 is arrived at by successive abstraction. From

zip (p:q) r = p: zip r q

we derive, by abstracting first from r,

zip (p:q) = p: |[zip r q *with* r=]| .

Here, the proper scope for r has been determined as follows: the placing of "*with* r=]|" is prescribed by our rule for λ-abstraction, the placing of the corresponding "|[" is determined by the rule that the scope coincides with the smallest syntactic unit containing all occurrences of the variable in question. Here its place reflects that functional application is left-associative, i.e. that zip r q is short for ((zip r) q).

On line 8, |[b _with_ b:?=y]| is no more than a complicated way of writing "hd y". The convention followed seems consistent — it is, in terms of number of characters needed, clumsy, but that is of no relevance here: we are not proposing a notation to be programmed in.

Things defined within a scope are significant within that scope. Our final result should have global significance. This observation presents the process of evaluation as exporting successive elements of x —which is a continued concatenation of integers— outside the scope of the local terminology. (We had already decided to choose our scopes as small as possible; exporting as much as possible from them is a natural extension. It can be claimed that we already did so, when we replaced

|[p : zip r q _with_ r=]|

by

p: |[zip r q _with_ r=]| .)

Evaluation of an expression can now be viewed as applying such reductions (i.e. semantics preserving transformations) that lead to export from a scope of values global with respect to that scope.

In this case we can do that by substitution. We have an expression of the form

$$|[\, x \ \underline{\text{with}}\ x = 1 : E(x)\,]| \ .$$

The substitution $x = 1 : w$ — meaningful because "1" has a meaning outside the scope of x — leads to the equivalent, but further evaluated expression

$$1 : |[\, w \ \underline{\text{with}}\ w = |[\, E(x)\ \underline{\text{with}}\ x = 1 : w \,]|\ ]| \ .$$

The inner bracket pair delineates the scope of x, the outer bracket pair delineates the scope of w.

Applying the substitution $x = 1 : w$ we get

```
1 : |[ w with w =                                        (0)
     |[ |[ zip with zip =                                (1)
              |[ p: |[ zip r q with r= ]| with p:q = ]|  (2)
         ]| x                                            (3)
        (|[ elf with elf =                               (4)
              |[a + |[b with b:? = y]| : elf y with a:y=]|   (5)
         ]| x) with x = 1 : w                            (6)
     ]|                                                  (7)
  ]|                                                     (8)
```

This is, to my feeling, not the moment to eliminate x

– by substituting 1:w for it – since, within its scope, x is referenced twice. The next thing to do is to unfold zip, i.e. replacing its outer occurrence by its definition and inserting its definition at its inner occurrence, i.e. replacing lines (1), (2), and (3) by

```
|[ |[ p : |[ |[ zip with zip =
                |[p:|[zip r q with r=]| with p:q=]|
            ]| r q with r=
         ]| with p : q =
   ]| x
```

Application of the above unfolded zip to x = 1: w leads to the elimination of the outer p:q, i.e the above can be simplified to

```
|[ 1 : |[ |[ zip with zip =
              |[p:|[zip r q with r=]| with p:q=]|
          ]| r w with r=
      ]|
```

We can now eliminate x by replacing its last occurrence by (1:w). The above being applied to the expression on 4-6, we also eliminate in the above the outer r, which is only referenced once. The combined result of these eliminations is

```
1 : |[ w with w = 1 :
      |[zip with zip =
          |[ p : |[ zip r q with r= ]| with p:q= ]|
      ]|
     (|[ elf with elf =
          |[a + |[b with b : ?=y]| : elf y with a : y= ]|
      ]| (1 : w)) w
   ]|
```

The next step is one we have seen before, viz substitution. This time we substitute something for w, say 1 : x. The result is

```
1 : 1 :
 |[ x with x =
     |[ |[ zip with zip =
             |[p: |[zip r q with r=]| with p:q=]|
        ]|
       (|[ elf with elf =
             |[a+ |[b with b:?=y]| : elf y with a:y=]|
        ]| (1 : w)) w  with w = 1 : x
     ]|
 ]|
```

Again, I have not felt the liberty of eliminating w, being referenced twice. The next step is to unfold zip again. In order to save writing I have performed the subsequent unfolding of elf as well.

```
1:1:                                                   (0)
|[ x with x =                                          (1)
  |[ |[ p:                                             (2)
        |[ |[zip with zip =                            (3)
               |[p: |[zip r q with r=]| with p:q=]|    (4)
           ]| r q with r=                              (5)
        ]| with p:q =                                  (6)
     ]|                                                (7)
    (|[a+ |[b with b:?=y]| :                           (8)
        |[ elf with elf =                              (9)
            |[a+ |[b with b:?=y]| : elf y with a:y=]|  (10)
        ]| y with a:y =                                (11)
     ]| (1:w)) w with w = 1: x                         (12)
  ]|                                                   (13)
]|                                                     (14)
```

Now we are ready to eliminate the outer a:y scope. lines 8-12 become then

```
(1+ |[b with b:?= w]|:
    |[ elf with elf =
        |[a+ |[b with b:?=y]| : elf y with a:y=]|
    ]| w)  w with w = 1: x
```

Now we can do several things. On account of "w=1:x" the top line of the above can be replaced by "(1+1:" and then by "(2:". We can also eliminate the outer p:q scope. They commute; then the outer r scope can be eliminated.

```
1:1:
|[ x with x = 2 :
   |[  |[ zip with zip =
            |[ p: |[ zip r q with r= ]| with p:q = ]|
       ]|  w
      ( |[ elf with elf =
            |[ a+ |[ b with b:?=y ]| : elf y with a:y = ]|
       ]| w) with  w = 1 : x
   ]|
]|
```

But now I get into problems. Taking the definitions of zip and elf for granted, the above can be represented as

```
1:1: |[ x with x = 2 : |[ zip  w (elf w)  with  w = 1 : x ]| ]|
```

and in a really demand-driven mode we would substitute x = 2 : y and we would get

```
1:1:2 :
  |[ y with  y =
        |[ |[ zip  w  (elf w )  with   w = 1 : x ]| with x = 2 : y ]|
  ]|
```

One way to proceed would be to eliminate the scope of x:

```
1:1:2:
 |[ y with y = |[ zip w  (elf w)  with  w = 1 : 2 : y ]| ]|
```

followed by an application of zip

$1:1:2:$
$|[\, y$ <u>with</u> $y = 1 : |[\, zip\ (elf\ w)\ (2:y)$ <u>with</u> $w = 1:2:y\,]|\ ]|$

followed by removal of w's single reference:

$1:1:2:$
$|[\, y$ <u>with</u> $y = 1 :\ zip\ (elf(1:2:y))\ (2:y)\ ]|$.

On the other hand, in a truly demand-driven way, we could have applied zip, using $w = 1 : x$

$1:1:2:$
$|[\, y$ <u>with</u> $y = 1 :$
$\quad |[\ |[\ zip\ (elf\ w)\ x$ <u>with</u> $w = 1 : x\ ]|$ <u>with</u> $x = 2:y\,]|$
$]|$

followed by a removal of w's single reference

$1:1:2:$
$|[\, y$ <u>with</u> $y = 1 : |[\, zip\ (elf(1:x))\ x$ <u>with</u> $x = 2:y\ ]|\ ]|$

I like that last form better, and consider the early removal of the scope of x — on account of "single occurrence" — a mistake. Let us, therefore, try only to substitute when forced to do so by a function application. Let us redo the game, but now in shorthand.

$\|[\ x\ \underline{\text{with}}\ x = 1 : \text{zip}\ x\ (\text{elf}\ x)\ ]\|$

The introduction $x = 1 : w$ leads to

$1 : \|[\ w\ \underline{\text{with}}\ w = \|[\ \text{zip}\ x\ (\text{elf}\ x)\ \underline{\text{with}}\ x = 1 : w\ ]\|\ ]\|$

Application of zip forces the first x to be substituted

$1 : \|[\ w\ \underline{\text{with}}\ w = 1 : \|[\ \text{zip}\ (\text{elf}\ x)\ w\ \underline{\text{with}}\ x = 1 : w\ ]\|\ ]\|$

The introduction of $w = 1 : y$ leads to

$1 : 1 : \|[\ y\ \underline{\text{with}}\ y = \|[\ \|[\ \text{zip}\ (\text{elf}\ x)\ w\ \underline{\text{with}}\ x = 1 : w\ ]\|\ \underline{\text{with}}\ w = 1 : y\ ]\|\ ]\|$

Application of elf is now possible, and the scope for x disappears

$1 : 1 : \|[\ y\ \underline{\text{with}}\ y = \|[\ \text{zip}\ (2 : (\text{elf}\ w))\ w\ \underline{\text{with}}\ w = 1 : y\ ]\|\ ]\|$

Now zip can be applied

$1 : 1 : \|[\ y\ \underline{\text{with}}\ y = 2 : \|[\ \text{zip}\ w\ (\text{elf}\ w)\ \underline{\text{with}}\ w = 1 : y\ ]\|\ ]\|$

Substituting $y = 2 : x$ we get

$1 : 1 : 2 :$
$\|[\ x\ \underline{\text{with}}\ x = \|[\ \|[\ \text{zip}\ w\ (\text{elf}\ w)\ \underline{\text{with}}\ w = 1 : y\ ]\|\ \underline{\text{with}}\ y = 2 : x\ ]\|\ ]\|$

Next we apply zip, using $w = 1 : y$ once

1:1:2:

$\|[\ x\ \underline{\text{with}}\ x = 1 : \|[\ \|[\ \text{zip}\ (\text{el}f\ w)\ y\ \underline{\text{with}}\ w = 1 : y\ ]\|\ \underline{\text{with}}\ y = 2 : x\ ]\|\ ]\|$

Etc. In this way the dilemma does not emerge. We need, of course, more and more identifiers, but that does not matter. We could use $f[k: k \geq 0]$; $f[k]$ stands then for $(+l)^k f$.

It is very well possible that the above example is much too simple. It suggests me, that garbage detection should not be a serious problem, and that should make me suspicious.

Note. I have not proved that with my last regime the dilemma does not arise; for the time being I am willing to believe it. (I am very willing to believe that the dilemma does not arise this way, for it is exactly the place where I got stuck a year ago). (End of Note.)

Plataanstraat 5
5671 AL NUENEN
The Netherlands

16 December 1981

prof. dr. Edsger W. Dijkstra
Burroughs Research Fellow