# To Friends


## Valentin Turchin, Andrei Nemytykh

**(Friends.to, Electronic Mail, The City University of New York, 04.04.1994, New York)**


Dear friends,

We report the creation of a self-applicable supercompiler.

It is still in the state of debugging, but we have run successfully a few very simple tests -- mostly with the "classical" function Fab:

```
Fab { e1 = <Fab1 () e1> }

Fab1 {
  (e1) 'a'e2 = <Fab1 (e1'b') e2>;
  (e1) sX e2 = <Fab1 (e1 sX) e2>;
  (e1) = e1;
   }
```

Here are three tests with this function. All tests were run in a completely automatic fashion; no adjustments have been done.

---

Test1. The MST scheme:

```
<Scp2 ..... >
        <Scp1 ……..... >
                <Fab e.1>
```

This is a correctness check. All work is done by Scp1, with Scp2 as a simple supervisor. Computation time: 1 sec.

The result is a program P2 which expects no inputs (function of no arguments) and outputs a program P1 for the computation of Fab. The program P1 is identical to the original program for Fab.

All I/O operations are in Refal graphs, which are hardly readable for anybody except ourselves, so we do not include them.

---

Test2 . The MST scheme:

<Scp2 ..... >

<Scp1 …….s.2 ….. >

<Fab | e.1>

Here the argument of Fab is s2 e1, and we raise s2 to become a free variable in the call of Scp1. Computation time was 3 sec.

The result is a program P2 which in Refal terms would be:

```
P1 {
  'a' = <Output P11>;
  s2 =  <Output P12>;
   }
```

where P11 is as the program for Fab1, except that the last sentence is

```
  (e1) = 'b'e1;
```

while P12's last sentence is:

```
  (e1) = s2 e1;
```

---

Test 3. The MST scheme:

```
<Scp2 ..... >
        <Scp1 ……..e.1.... >
                <Fab ..|...>
```

Here we raise the whole argument e1. Computation time 23 sec.

The result in the Refal form is:

```
Fab { e1 = <Fab1 () e1> }

Fab1 {
  (e1) 'a'e2 = <Fab1 (e1'b') e2>;
  (e1) sX e2 = <Fab1 (e1 sX) e2>;
  (e1) = (A (e1)'e'0);
  e1 = Z;
    }
```

where (A (e1)'e'0) is the Refal graph notation for a function that outputs e1, and Z is the notation of the situation 'recognition impossible'; it is a metacode representation of the situation when the main level program goes into an abnormal stop.

It is too early to speak of real practical applications of self-applicable supercompilation. Debugging and improvements are in order. But a self-applicable supercompiler does, finally, exist.

There are two major advancements in the present Scp as compared to the earlier models.

The first: MS jumps from one level to another, which allow to avoid interpretation when a direct computation is possible. Second: the concept of the ELEVATION of a free variable is introduced -- it is of vital importance for self-application. Also, the restriction of the input language to flat Refal helped a lot. A detailed description of all these features is under preparation.

The supercompiler itself, i.e. Scp2, is for flat Refal programs.

It is written in the extended Refal; its volume: 302 sentences.

It was manually translated into strict Refal; the volume became 347 sentences. Then it was automatically translated into a flat form (499 sentences), which then was used as Scp1.

Best wishes

Valentin Turchin

Andrei Nemytykh