

THE COMPUTER REVOLUTION IN PHILOSOPHY (1978)

Aaron Sloman

[Book contents page](#)

This postscript is also available [in PDF format here](#).

POSTSCRIPT

DO WE NEED A HIERARCHY OF METALANGUAGES?

It is widely believed that the work of Russell and Tarski has established that we need a hierarchy of distinct metalanguages, if we wish to use concepts like 'true', 'refers to' and other semantic concepts. The argument is based on such facts as that a sentence like

"This statement is not true"

must be false if it is true, and true if it is false. This, and other versions of the liar paradox, and related paradoxes, can be used to show that if the law of the excluded middle is correct (every statement is either true or not true) then contradictions can be generated in languages which 'contain their own metalanguage'.

Many philosophers and logicians have inferred from this that only a hierarchy of distinct metalanguages provides a safe framework for precise and rigorous theorising in science or mathematics. I have argued against this in my 1971 paper ('Tarski Frege and the Liar Paradox'), but would now like to illustrate the way in which precise, rigorous, and widely used programming languages generate similar paradoxes in a very natural and easily understood way.

In the AI programming language Pop-11[1] this is how you can define a little program[2] which tests whether a list evaluates to **true**:

```
define ISTRUE(list);
  pop11_compile(list) = true
enddefine;
```

So (using the procedure **pr** to print the result produced by the procedure **ISTRUE**:

```
pr(ISTRUE( [ 8 > 5 ] ));
```

prints out:

```
<true>
```

since 8 is bigger than 5, whereas:

```
pr(ISTRUE( [ isinteger("cat") ] ));
```

prints out:

```
<false>
```

because the *word* "cat" in is not an *integer*.

We can declare a variable name S, thus:

```
vars S;
```

Now assign to it a list which asserts that what S says is not true:

```
[not(ISTRUE(S))] -> S;
```

If we now ask the Pop-11 system to check whether S is true and print out the result, thus:

```
pr(ISTRUE(S));
```

the system grinds to a halt and prints out an error message, because of the 'infinite recursion' generated, i.e. it runs out of work-space trying to tell if S is true, which requires working out if S is true, which requires working out if S is true ...

So we have no contradiction, just a non-terminating process, which happens to be stopped when memory runs out. (In some implementations of this sort of language, so-called 'tail-recursion optimisation' might be used, which would prevent memory running out and the program would run forever.)

There is a *contradiction* only if you assume that every well-formed sentence (including S) must have a definite truth-value, a comm prejudice, for which there is no foundation.

We can do a similar demonstration with Russell's paradox. Pop-11, like many other programming languages, has built in procedures which work as predicates, producing a truth value when applied to an argument, e.g. **isinteger**, **isword**, **isprocedure**. These are all objects of type procedure, in Pop-11. So:

```
pr(isinteger(3));
<true>
pr(isprocedure(isinteger));
<true>
pr(isinteger(isinteger));
<false>
pr(isprocedure(isprocedure));

<true>
```

We can define a new procedure, called RUSSELL, as follows:

```
define RUSSELL(f);
  not(f(f))
enddefine;
```

This defines RUSSELL as a predicate. The command

```
pr(RUSSELL(isprocedure));
```

causes `isprocedure` to be applied to itself, yielding

```
<true>
```

which is then negated, and

```
<false>
```

is printed out.

Similarly

```
pr(RUSSELL(isinteger));
```

causes

```
<true>
```

to be printed out, since **isinteger** is a procedure, not an integer. So the procedure is perfectly well defined, and generally works.

However, execution of the command

```
pr(RUSSELL(RUSSELL));
```

cannot terminate until it has checked whether **RUSSELL** applied to **RUSSELL** yields **true** or **false**, which in turn needs the same check. So once again the system starts infinite recursion, and eventually grinds to a halt with an error message if memory runs out.

Far from showing a need for a hierarchy of distinct metalanguages, this merely illustrates the fact that a well-formed expression with a clear sense, (e.g. a clearly defined evaluation procedure), need not determine a definite reference (e.g. because the procedure never terminates). This is inevitable in any general purpose programming language. No wonder it is a feature of natural languages.

End Notes

[1] In the original (1978) version of the book, the programming examples used the syntax of the language POP2 (Burstall et al 1973). In this version (Sept 2001) I have changed the syntax to that of Pop-11, which is now freely available from this site:

<http://www.cs.bham.ac.uk/research/poplog/freepoplog.html>

[2] The Pop-11 procedure **pop11_compile** when applied to a list of program text items, compiles and executes the text.

Last Updated: 4 Jun 2007

[Book contents page](#)

[Next: Bibliography](#)