# The halting problem illustrated with C functions

Frédéric Lang

September 7, 2015

In this problem, assume that there exists a C function terminate() with the following interface:

```
boolean terminate (void *p);
/*
 * Precondition: p is a pointer to a parameter-free function
 * Postcondition: return true if and only if (*p)() terminates
 * Property: terminate (*p) terminates on any input p
 */
```

We are going to show that such a terminating terminate() function cannot exist[1]. To this aim, we define the following four functions skip(), loop(), loop_if_terminate(), and self_loop_if_terminate():

```
void skip () {}    /* do nothing and terminate */

void loop () {
     for ( ; ; ); /* do nothing and never terminate */
}

boolean loop_if_terminate (void *p) {
     boolean b = terminate (p);
     if (b) loop (); else return b;
}

boolean self_loop_if_terminate () {
     return loop_if_terminate (&self_loop_if_terminate);
}
```

**Questions:**

1. Consider the parameter-free functions f01() to f12(), defined in Figure 1. How should calls to these functions behave? Indicate whether the function should loop indefinitely or terminate, and in the latter case, the result it should return. You do not need to justify your answer.
2. Assume that the call self_loop_if_terminate () does not terminate. How can you contradict that asumption?

[1] Note that if such a function cannot exist for parameter-free functions, then it cannot exist at all for more general functions, since general functions include parameter-free functions.

```
boolean f01 () { return terminate (&loop); }
boolean f02 () { return terminate (&skip); }
boolean f03 () { return loop_if_terminate (&loop); }
boolean f04 () { return loop_if_terminate (&skip); }
boolean f05 () { return terminate (&f01); }
boolean f06 () { return terminate (&f02); }
boolean f07 () { return terminate (&f03); }
boolean f08 () { return terminate (&f04); }
boolean f09 () { return loop_if_terminate (&f01); }
boolean f10 () { return loop_if_terminate (&f02); }
boolean f11 () { return loop_if_terminate (&f03); }
boolean f12 () { return loop_if_terminate (&f04); }
```

Figure 1: Various functions

3. Assume that the call self_loop_if_terminate () does terminate. How can you contradict that asumption again?

4. What can you conclude about the existence of a function having the behaviour intended for terminate()?

5. Is C Turing complete? How would you prove this fact?

6. In theory, can any C program be emulated by a Turing Machine?

7. How do you come from your conclusions at questions 4, 5, and 6, to a famous result in theoretical computer science?