Programming Language Semantics and Compiler Design

(Sémantique des Langages de Programmation et Compilation)

Exercise: The Halting Problem

Frédéric Lang & Laurent Mounier
firstname.lastname@univ-grenoble-alpes.fr
Univ. Grenoble Alpes, Inria,
Laboratoire d'Informatique de Grenoble & Verimag

Master of Sciences in Informatics at Grenoble (MoSIG)
Master 1 info

Univ. Grenoble Alpes - UFR IM^{2}AG
www.univ-grenoble-alpes.fr — im2ag.univ-grenoble-alpes.fr

Goal of this exercise

We want to answer the following question:

In an arbitrary programming language, is it possible to define a procedure that takes as input any representation of a program P and provides as output a Boolean which says whether P terminates (**tt**) or not (**ff**)?

Note that such a function should provide a result (i.e., terminate) on any input!

To answer this question, we will first:

- define the minimal characteristics of an arbitrary programming language that all general-purpose programming languages have
- assume that it is possible to write such a function that we will call terminate
- study how terminate should behave when taking as input the representation of various functions

The programming language

We consider an arbitrary programming language, which we call **L**.

We require **L** to have:

- Functions that may have input parameters and may return a result.
- Data structures enabling an encoding of **L**'s own programs (e.g., abstract syntax trees, binary code stored in files or character strings, etc.).

  If **L** is typed, we call fun the main type of this representation.

Functions

It must be possible to write at least the following functions
(types can be ignored if **L** is not a typed language):

- A nullary function "ftrue () : bool", which always returns the value **tt**.
- A nullary function "loop () : bool", which loops forever (i.e., never returns).
- A unary function "loop_if (*b* : bool) : bool", which loops forever if *b* is **tt** and returns *b* otherwise.

Exercise 1

Write those function in your favorite programming language, or in **While** extended with function definitions.

Solution of exercise 1

- In C:

```
bool ftrue () { return true; }
bool loop () { for ( ; ; ); return true; }
bool loop_if (bool b) { if (b) return loop (); return b; }
```

- In **While** extended with function definitions and Boolean results:

```
ftrue () : bool = return true
loop () : bool = while true do skip od; return true
loop_if (b : bool) : bool = if b then return loop () else return b fi
```

Representation of **L** functions using **L** data structures

For a function F written in the language **L** we use the meta-notation

$$[\![F]\!]$$

to denote the **L** data structure (of type fun) representing F.

Example 1

$[\![\text{loop\_if}]\!]$ denotes the **L** data structure representing function loop_if .

The terminate function

We assume the existence in **L** of the function:

$$\text{terminate}(f : \text{fun}) : \text{bool}$$

so that terminate($[\![F]\!]$) evaluates to **ff** if function F may loop forever on some input, and to **tt** otherwise.

We also define in **L** the following two nullary functions:

- loop_if_terminate (f : fun) : bool = loop_if (terminate(f))
- self_loop_if_terminate () : bool =
  loop_if_terminate ($[\![$self_loop_if_terminate$]\!]$)

Evaluation of calls to terminate and other functions

Exercise 2

What does terminate ($[\![F]\!]$) return for F ranging among the following functions:

f01 () : bool = terminate ($[\![$loop$]\!]$)
f02 () : bool = terminate ($[\![$ftrue$]\!]$)
f03 () : bool = loop_if_terminate ($[\![$loop$]\!]$)
f04 () : bool = loop_if_terminate ($[\![$ftrue$]\!]$)
f05 () : bool = terminate ($[\![$f01$]\!]$)
f06 () : bool = terminate ($[\![$f02$]\!]$)
f07 () : bool = terminate ($[\![$f03$]\!]$)
f08 () : bool = terminate ($[\![$f04$]\!]$)
f09 () : bool = loop_if_terminate ($[\![$f01$]\!]$)
f10 () : bool = loop_if_terminate ($[\![$f02$]\!]$)
f11 () : bool = loop_if_terminate ($[\![$f03$]\!]$)
f12 () : bool = loop_if_terminate ($[\![$f04$]\!]$)

Solution of Exercise 2

f01 () = false
f02 () = true
f03 () = false
f04 () loops forever
f05 () = true
f06 () = true
f07 () = true
f08 () = false
f09 () loops forever
f10 () loops forever
f11 () loops forever
f12 () = false

First hypothesis: self_loop_if_terminate () loops forever

Exercise 3

Assume that the call self_loop_if_terminate () loops forever. What can you conclude from this function's definition?

Solution of Exercise 3

The call self_loop_if_terminate () loops forever if and only if:

- loop_if_terminate (⟦self_loop_if_terminate⟧) loops forever, by definition of self_loop_if_terminate, i.e.,
- loop_if (terminate(⟦self_loop_if_terminate⟧)) loops forever, by definition of loop_if_terminate, i.e.,
- terminate(⟦self_loop_if_terminate⟧) is true, by definition of loop_if, i.e.,
- self_loop_if_terminate always terminate, by definition of terminate.

Therefore, self_loop_if_terminate () loops forever if and only if it always terminate. This is contradictory. Thus, if terminate exists, then the function call self_loop_if_terminate () terminates.

Second hypothesis: self_loop_if_terminate () terminates

Exercise 4

Assume that the call self_loop_if_terminate () terminates. What can you conclude from this function's definition?

Solution of Exercise 4

The call self_loop_if_terminate () terminates if and only if:

- loop_if_terminate (⟦self_loop_if_terminate⟧) terminates, by definition of self_loop_if_terminate , i.e.,
- loop_if (terminate (⟦self_loop_if_terminate⟧)) terminates, by definition of loop_if_terminate , i.e.,
- terminate (⟦self_loop_if_terminate⟧) is false, by definition of loop_if , i.e.,
- self_loop_if_terminate may loop forever, by definition of terminate .

Since self_loop_if_terminate is a deterministic function that does not take any input, this means that the call self_loop_if_terminate () loops forever (it is the only possible call to function self_loop_if_terminate).
Therefore, self_loop_if_terminate () terminates if and only if it loops forever.
This is contradictory. Thus, if terminate exists, then the function call self_loop_if_terminate () loops forever.

Conclusion

Exercise 5

What can you conclude from Exercises 4 and 5?

Conclusion

Exercise 5

What can you conclude from Exercises 4 and 5?

We made a single assumption: function terminate exists. Building on this, we proved that the function call self_loop_if_terminate () both terminates and loops forever.

This contradiction means that the assumption is not valid.

Therefore, it is not possible to write such a function in any programming language having the desired functionalities for **L**. This includes your favorite programming language.

Said differently, it is not possible to prove automatically that a program terminates.

This confirms Alan Turing's result: there is no procedure to determine whether a Turing machine halts. This is known as undecidability of the halting problem. This is not a suprise as any program can be simulated by a Turing machine and any Turing machine can be simulated by a program (a property of programming languages known as Turing completenes). If we had shown that terminate exists, we would have contradicted Turing's result.