

Exercise 1. r.e. languages

Which of the following properties of r.e. sets are themselves r.e.?

1. L contains at least two strings.
2. L is infinite.
3. L is a context-free language.
4. $L = L^*$

Exercise 2. Halting problem

Show that it is undecidable whether a TM halts on all inputs.

Exercise 3. Primitive recursion

Ackerman's function is defined as follows.

$$\begin{aligned}A(0, x, y) &= s(y) \\A(1, x, 0) &= x \\A(2, x, 0) &= 0 \\A(n, x, 0) &= 1 \quad \text{for } n \geq 3 \\A(s(n), x, s(y)) &= A(n, x, A(s(n), x, y))\end{aligned}$$

1. Show that for any fixed n , The function $f(x, y) = A(n, x, y)$ is primitive recursive.
2. Show that A is not primitive recursive.

Exercise 4. Self-reference

Using your favorite programming language (or any programming language for that matter), define a program that prints itself out. Your program should be as short as possible.

Exercise 5. Partial recursive functions

Let σ be an arbitrary total recursive function mapping indices of partial recursive functions to indices of partial recursive functions. Give an algorithm to enumerate an infinite set of fixpoints of σ ; that is, infinitely many i 's such that $\varphi_i(y) = \varphi_{\sigma(i)}(y)$ for all y .

Exercise 6. Laboratory

Write Ackerman's function in OCaml. Your program should be called `ackerman`, and it should take three command-line arguments, and print the result. Here is an example:

```
<jyh:kenai 10>./ackerman 0 5 7
Result: 8
<jyh:kenai 11>./ackerman 1 5 7
Result: 12
<jyh:kenai 12>./ackerman 2 5 7
Result: 35
<jyh:kenai 13>./ackerman 3 5 7
Result: 78125
<jyh:kenai 14>./ackerman 4 5 7
Fatal error: exception Stack_overflow
Exit 2
```

Here is the code sketch. You just need to implement the ackerman function.

```
open Num
open Printf

let zero = Int 0
let one = Int 1
let two = Int 2

let rec ackerman n x y =
  raise (Failure "not implemented")

let main () =
  let n = num_of_string Sys.argv.(1) in
  let x = num_of_string Sys.argv.(2) in
  let y = num_of_string Sys.argv.(3) in
  let z = ackerman n x y in
  printf "Result: %s\n" (string_of_num z);
  flush stdout

let _ = main ()
```

You will need to use the OCaml Num library, documented in the OCaml manual. To compile the program, use the following command:

```
% ocamlc -warn-error A nums.cma -o ackerman ackerman.ml
```

Exercise 7. Laboratory

Write a program in OCaml that evaluates a simple arithmetic expression. Your program should be called `expr`. The simple expressions have the form:

$$e ::= \text{NUMBER } op \text{ NUMBER}$$
$$op ::= + | - | * | /$$

The three arguments are given on the command line. Here is a sample run:

```
<jyh:kenai 19>./expr 1 + 2
Result: 3
<jyh:kenai 20>./expr 1 * 2
Fatal error: exception Failure("num_of_string")
Exit 2
<jyh:kenai 21>./expr 1 "*" 2
Result: 2
<jyh:kenai 22>./expr 5 "*" 7
Result: 35
<jyh:kenai 23>./expr 5 "/" 7
Result: 5/7
<jyh:kenai 24>./expr 5 "-" 7
Result: -2
```

To get started, use the code sketch from Ackerman's function, but replace the `ackerman` function with a function to evaluate the arithmetic expression.