How ChatGPT works as A Pedagogical tool

Jaylene Zhang

Our motivation

- LLM are highly successful for many applications

- In particular, LLMs have been successfully used for program synthesis/generation in java/python/…..

Our goal

- Evaluate LLM for programming in the setting of teaching statically typed functional programming, in particular OCaml

Which helps us learn …

- the capability of different LLM in writing programs
- how students could use LLM effectively in a course
- how students could use LLM to solve programming exercises, which gives insights to instructors how to best phrase programming exercises

ChatGPT to generate HW codes given HW description

***Prompt**: homework description with function signature*
***Output from the model**: solution code*

e.g.
The Lucas numbers form an integer sequence similar to the Fibonacci sequence. They are defined by the following (recursive) equations:

L(0) = 2
L(1) = 1
L(n) = L(n-1) + L(n-2)

Implement the function lucas : int -> int such that lucas n calculates L(n). Your implementation must be tail-recursive. Hint: use an inner helper function with additional parameters to carry forward the values for L(n-1) and L(n-2).

```
(* TODO: Implement a tail-recursive helper lucas_helper. *)
let rec lucas_helper params =
  raise NotImplemented

(* TODO: Implement lucas that calls the previous function. *)
let lucas n =
  raise NotImplemented
```

ChatGPT to generate HW codes

We record the maximum grade for each homework question among 5 trials:

HW with 100% = 18 / 31
HW with more than 80% = 3 / 31
HW with 0% = 8 / 31
other = 2 / 31

Overall homework grade: ~ 70%

Bad use of ChatGPT

Q1. Overwriting vs overshadowing

Q: Give an example of overwriting and explain how overwriting is different from for example overshadowing.

A: Overwriting happens when a variable is re-assigned a new value in the same scope, replacing the previous value. This means the original value is completely replaced by the new one.

Example of Overwriting: (This is actually overshadowing!)

```
let x = 5

let x = 10  (* This overwrites the value of x from 5 to 10 *)
```

In this case, the second assignment to x overwrites the original value, so after the second assignment, x will hold the value 10.

In contrast, overshadowing occurs when a variable in a more specific scope (e.g., a function) has the same name as a variable in an outer scope, but does not directly overwrite it. Instead, the inner variable hides or shadows the outer variable within its scope.

Some statistics

Among all the submitted exams (around 200) …

- 45 had deductions of more than 2 points (out of a total of 8 points).
- 36 mentioned something related to “outer scope” or “inner scope.”

Q2. Reference

Q: What is the result of evaluating the expression below using OCaml's call-by-value strategy? – Show the binding stack to explain your reasoning and answer.

```
let x = 3 in
let r = ref (2 + x) in
let f = (let y = x * !r in
let r = ref 4 in fun u -> u + x + y + !r ) in
let x = 5 in
let y = 10 in
r := 2 ; f (!r * y)
```

A:

Binding Stack (call-by-value evaluation):

Outer Scope:

x = 3

r = ref 5

Function Definition (f):

y = 15

r = ref 4 (local to f)

Q2. Reference

Q: What is the result of evaluating the expression below using OCaml's call-by-value strategy? – Show the binding stack to explain your reasoning and answer.

```
let x = 3 in
let r = ref (2 + x) in
let f = (let y = x * !r in
let r = ref 4 in fun u -> u + x + y + !r ) in
let x = 5 in
let y = 10 in
r := 2 ; f (!r * y)
```

A:

Inner Scope:
x = 5 (shadows outer x)
y = 10
r = ref 2 (updated reference)

Evaluation of f (!r * y):
u = 20
x = 5 (from the inner let)
y = 15 (calculated earlier)
!r = 4 (local to f)

Final result = 20 + 5 + 15 + 4 = 44.
Thus, the final evaluated result is 44.

(The final result is 42 since x should be 3)

- Incorrect solutions due to lack of understanding
- Limited debugging capabilities
- Inconsistant response

Good use of ChatGPT

Tail-recursive function

Q: Explain in your own words when a function is tail-recursive using examples.

A: A function is tail-recursive if the recursive call is the last operation in the function. This means that the result of the recursive call is directly returned without any further computation after it. Tail-recursive functions are more memory-efficient because they allow the compiler or interpreter to optimize the recursion into an iterative loop, preventing the creation of new stack frames for each recursive call.

Example of Tail-Recursive Function:

```
let rec factorial_tail_rec n acc =
  if n <= 0 then acc
  else factorial_tail_rec (n - 1) (n * acc)
```

Here, factorial_tail_rec is a tail-recursive function. The recursive call factorial_tail_rec (n - 1) (n * acc) is the last operation in the function. After the recursive call, no further computation is needed, and the value of acc is returned directly. This allows the function to be optimized into a loop by the compiler.

Use ChatGPT to generate practice problems

Variations from 100 exercises in OCaml

(https://ocaml.org/exercises?difficulty_level=intermediate)

Original: Flatten a nested list structure.
Variation: Flatten a nested list structure and order the list

Original: Eliminate consecutive duplicates of list elements.
Variation: Modify the compress function so that it returns not only the list with consecutive duplicates removed but also a count of how many times each element was consecutively repeated.

Original:Rotate a list N places to the left.
Variation: Modify the rotate function to take an additional boolean parameter, right, which, if true, rotates the list N places to the right instead of the left.

Original:Pack consecutive duplicates of list elements into sublists.
Variation: Rewrite the pack function without using List.rev to reverse the list at the end. Instead, construct the list in the correct order using only direct recursion.

Original: Replicate the elements of a list a given number of times.
Variation: Rewrite the replicate function without using List.rev or List.fold_left, constructing the final list in the correct order directly through recursion.

Original: Drop every N'th element from a list.
Variation: Rewrite the drop function to return both the resulting list with every N'th element removed and a list of the dropped elements. This could be useful for tracking what was removed in addition to what remains.

… …

Example1 Eliminate Duplicates

Original question: Eliminate consecutive duplicates of list elements.

Variation: Modify the compress function so that it returns not only the list with consecutive duplicates removed ***but also a count of how many times each element was consecutively repeated.***

Example1 Eliminate Duplicates

Q: Modify the compress function so that it returns not only the list with consecutive duplicates removed ***but also a count of how many times each element was consecutively repeated.***

```
let compress_with_counts lst =
  let rec aux acc count = function
    | [] -> List.rev acc
    | [x] -> List.rev ((x, count + 1) :: acc)
    | a :: (b :: _ as t) ->
        if a = b then aux acc (count + 1) t
        else aux ((a, count + 1) :: acc) 0 t
  in
  aux [] 0 lst;;
```

- Generate variations and extensions of standard problems, encouraging students to explore beyond the basics.

- Such problems can be *self-tested* using LearnOCaml immediately.

- Reduce the effort required from instructors to create new examples.