TA Evaluation Problem Set

This problem set contains four questions. If something is unclear, please make reasonable assumptions and state them in your solution. **Submit your answers as a single PDF document by email to the instructors by January 7**.

1 Mandatory Induction Question

The weak form of the Principle of Mathematical Induction states that:

$$P(0) \land (\forall n \in \mathbb{N}.(P(n) \to P(n+1))) \to \forall n \in \mathbb{N}.P(n) \tag{1}$$

The Cauchy form of the Principle of Mathematical Induction states that:

$$P(1) \land (\forall n \in \mathbb{N}.(P(n) \to P(2n))) \land (\forall n \in \mathbb{N}.(P(n+1) \to P(n))) \to \forall n \in \mathbb{N}.P(n) \tag{2}$$

Prove that Equation 1 is equivalent to Equation 2.

2 Programming

Design and implement a Python program to determine whether a given Sudoku puzzle has a *unique* solution. Your program should verify the uniqueness of the solution and include a proof of the correctness of the algorithm used.

3 Debugging

The following OCaml code contains a semantic (logical) error. Your task is to:

1. Explain what the code is *intended* to do,
2. Identify the semantic (logical) error in the code,
3. Provide an example input that triggers the error, and
4. Write a corrected version of the code.

```
let rec u x y =
  match (x, y) with
  | ([], _) -> y
  | (_, []) -> x
  | (h1 :: t1, h2 :: t2) ->
      if h1 <= h2 then h1 :: (u t1 y) else h2 :: (u x t2)

let s l =
  let l' = (List.length l) / 2 in
```

```
  let rec t n m =
    match (n, m) with
    | (0, _) -> []
    | (_, []) -> []
    | (c, h :: t) -> h :: (t (c - 1) t)
  in
  let rec d p q =
    match (p, q) with
    | (0, r) -> r
    | (_, []) -> []
    | (z, _ :: r) -> (d (z - 1) r)
  in
  ((t l' l), (d l' l))

let rec ms z =
  match z with
  | [] -> []
  | [_] -> z
  | _ ->
      let (a, b) = (s z) in
      u (ms a) (ms b)
```

4 Grading Exercise

Design a rubric for the following, grade the two submissions, and provide constructive feedback and criticism as you would in a course.

Problem: Let n-bonacci number F_n be defined as follows:

- For all $k \in \mathbb{N}, k < n$, $F_n(k) = 1$
- For all $k \geq n$, $F(k) = \sum_{i=0}^{n-1} F_n(k-i)$

This is a generalization of Fibonacci numbers. Write a pseudo-code to *efficiently* compute $F_n(2n)$, the $2n^{\text{th}}$ n-bonacci number.

Solution A:

```
function n_bonacci_A(n, k):
    if k < n:
        return 1

    terms = [1] * n

    for i from n to k:
        next_term = 0
```

```
        for j from 0 to n - 1:
            next_term += terms[j]

        for j from 0 to n - 2:
            terms[j] = terms[j + 1]
        terms[n - 1] = next_term

    return terms[n - 1]
```

Solution B:

```
function n_bonacci_B(n, k):
    memo = array of size k initialized to -1

    function helper(x):
        if memo[x] != -1:
            return memo[x]
        if x <= n:
            return 1

        sum = 0
        for i from 1 to n:
            sum += helper(x - i)

        memo[x] = sum
        return sum

    return helper(k)
```