UNIFORM COMPUTATION

I. LOOPS AND INFINITY

A. Turing machines

B. NAND-TM

NAND-TM = NAND-CIRC + loops + arrays

1. We add a special integer valued variable i. All other variables in NAND-TM will be Boolean valued (as in NAND-CIRC).
2. We add arrays to the language by allowing variable identifiers to have the form Foo[i] with i being the special integer-valued variable mentioned above. Foo is an array of Boolean values, and Foo[i] refers to the value of this array at location equal to the current value of the variable i.
3. We use the convention that arrays always start with a capital letter, and scalar variables (which are never indexed with i) start with lowercase letters. Hence Foo is an array and bar is a scalar variable.
4. The input and output X and Y are now considered arrays with values of zeroes and ones.
5. We add a special MODANDJUMP instruction that takes two boolean variables a, b as input and does the following:
   - a=1,b=1
   - a=0,b=1
   - a=1,b=0
   - a=0,b=0

Theorem 1 *Turing machines and NAND-TM programs are equivalent. For every $F : \{0,1\}^* \to \{0,1\}^*$, F is computable by a NAND-TM program P if and only if there is a Turing Machine M that computes F.*

II. EQUIVALENT MODELS OF COMPUTATION

A. RAM machines and NAND-RAM

B. Lambda calculus

We start with "basic expressions" that contain a single variable such as x or y and build more complex expressions using the following two rules:

1. Application: If e and e' are λ expressions, then the λ expression (ee') corresponds to applying the function described by e to the input e'.
2. Abstraction: If e is an expression and x is a variable, then the λ expression $\lambda x.(e)$ corresponds to the function that on any input z returns the expression $e[x \to z]$ replacing all (free) occurrences of x in e.

Definition 1 (λ expression) *A λ expression is either a single variable identifier or an expression that is built from other expressions using the application and abstraction operations.*

Definition 2 (Equivalence of λ expressions) *Two expressions are equivalent if they can be made into the same expression by repeated applications of the following rules:*

1. *Evaluation (β reduction): The expression $(\lambda x.exp)exp'$ is equivalent to $exp[x \to exp']$.*
2. *Variable renaming (α conversion): The expression $\lambda x.exp$ is equivalent to $\lambda y.exp[x \to y]$.*

Example: DOUBLE

$$DOUBLE\ f = \lambda f.(\lambda x.f(fx)) \tag{1}$$

C. The "ENHANCE" λ Calculus

The enhanced λ calculus includes the following set of objects and operations:

- Boolean constants and IF function: The enhanced calculus has the constants 0 and 1 and the IF function such that for every $cond \in \{0, 1\}$ and λ expressions a, b, IF cond a b outputs a if $cond = 1$ and outputs b if $cond = 0$.
- Pairs: We have the function PAIR such that $PAIR\ x\ y$ returns the pair (x, y) that holds x and y.
- Lists and strings: Using PAIR we can also construct lists. The idea is that $PAIR\ a\ L$ corresponds to the list obtained by adding the element a to the beginning of a list L. A string is simply a list of bits.
- List operations: MAP, REDUCE, and FILTER.
- Recursion: if we have a function F taking two parameters me and x, then RECURSE F will be the function f taking one parameter x such that $f(x) = F(f, x)$ for every x.

1. *Enhanced λ expressions*