Computability and Complexity - Assignment 2

Mikkel Willén
`mw@di.ku.dk`
Collaborators: Caroline Amalie Kierkegaard, `qlj556`

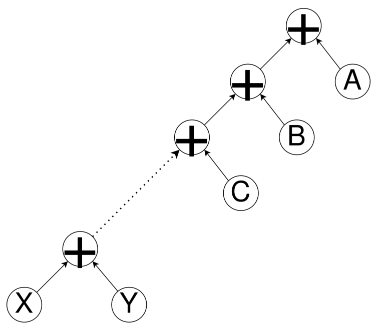
March 15, 2024

Task 1

To show that a language $L \in \mathbf{P/poly}$, we need to show, that there exists a polynomial-sized circuit family $\{C_n\}$ such that for every input of size n, the circuit C_n decides whether the input is in L.
Since L is sparse, we know $|L \cap \{0,1\}^n| \leq p(n)$, which means the language L contains less than $p(n)$ strings of the length n for all $n \in \mathbb{N}^+$. This means we can create a circuit C_n where $|C_n| \leq p(n)$ for all $n \in \mathbb{N}^+$. This construction satisfies the definition of a language being in $\mathbf{P/Poly}$ and thus we have shown that if L is sparse, then $L \in \mathbf{P/Poly}$.

Task 2

Assuming that `XOR` is modulo-2 addition, we would create the Boolean circuit:



Where there are $n-1$ `XOR` operators, with the `XOR` operators being a constant number of boolean gates each (namely a $\neg$, a $\wedge$ and a $\vee$). This can be constructed in linear time.

Task 3

To show that the function is computable in $O(n)$ space, we create an algorithm, that performs the multiplication.

The general idea is:

- Initialize a result variable to store the product
- Iterate over each bit of the multiplier
  1. if the current bit is 1, we add the multiplicand to the result variable
  2. Shift the multiplicand left by one

- return the result variable

The algorithm will look like the following

Algorithm 1: `BinaryMultiplication`

```
Data: $A$ of length $n$, $B$ of length $n$
Result: The product of A and B
result ← 0;
i ← 0;
while $i \leq n$ do
    if $B[i] == 1$ then
        result ← result + A
    end if
    Bitshift A left by 1
    $i \leftarrow i + 1$;
end while
Return result
```

Since the result can atmost have length $2n$ and since we only have to store A, B and the result we only need $O(n)$ space.

Task 4

We construct a Turing Machine M which checks whether a given input is in the language L. M works as follows: We first check if the first number is 1 and then followed by a #. If it is not 1 we reject. If it is 1 and there is no hashtag, we accept. Else we move on. If there is a hashtag we enter into a loop. In the loop, we check if the next number is all ones and then end of input. We save this as a boolean variable with the value true, if it is all ones and end of input, and false otherwise. We then subtract the previous number from this number with ones complement, meaning we look at the last bits of both numbers, flip the bit of the subtrahend and add them together. We save a bit with the potential carryover. Then we do the same for the next bit and also add the carryover. When there are no more bits in the number, we should have a one in the carryover. If there is not, we reject. If there is and the boolean bit is true, we accept. Else we move on to the next number.

With this, we only need to store a constant number of bits. A carryover, and exit truthvalue and the 1 bit for each of the numbers we are looking at.

Task 5

To show that SpaceBoundTM is **PSPACE**-complete, we first need to show that we can verify a witness in polynomial space.

We do this by creating another Turing machine, that simulates M on the input x and 1^n. If the turing machine every use more than n space, we reject. This can be done with $O(n+1)$ space, since as soon as we use more than n space, we reject.

We now have to show a polynomial space reduction from another **PSPACE**-complete problem to SpaceBoundTM.

Given an arbitrary language $L \in$ **PSPACE**, there exists a polynomial-space Turing machineM_L that decides L. We can construct a reduction from L to SpaceBoundTM as follows. Given input x, we want to decide whether x belongs to L. We construct a new input $\langle M_L, x, 1^{|x|}$ for SpaceBoundTM, where M_L is the description of the polynomial-space Turing machine that decides L. We then run SpaceBountTM on the input, and if SpaceBoundTM accepts, x is in the language L and if it rejects it is not. So we have that if x belongs to L, then there exists a computation path of M_L on x that accepts within polynomial space, and if x does not belong to L, then for all compuation paths of M_L on x none of them will accept within polynomial space.

Since the reduction from L to SpaceBountTM runs in polynomial time and correctly maps instances of L to instances of SpaceBoundTM, and since SpaceBoundTM is in **PSPACE**, this proves that SpaceBoundTM is **PSPACE**-complete.

Task 6

To show that $\Sigma_2\mathbf{SAT}$ is Σ_2^p-complete under polynomial-time reductions, we need to demonstrate two thing. First that $\Sigma_2\mathbf{SAT}$ is in Σ_2^p, and that every language $L \in \Sigma_2^p$ can be reduced to $\Sigma_2\mathbf{SAT}$ in polynomial time.

To show that $\Sigma_2\mathbf{SAT}$ is in Σ_2^p we need to exhibit a polynomial-time predicate for $\Sigma_2\mathbf{SAT}$. $\Sigma_2\mathbf{SAT}$ consts of true totally quantified Boolean formulas on the form:

$$\exists y_1 \exists y_2 \ldots \exists y_n \forall z_1 \forall z_2 \ldots \forall z_m F(y_1, \ldots, y_n, z_1, \ldots, z_m)$$

Given a formula Φ in $\Sigma_2\mathbf{SAT}$ and a satisfying assignment, we need to verify whether the assignment satisfies Φ. We start by non-deterministically guess assignments for the existential quantifiers $y_1, y_2, \ldots, y_n$. Then for each guessed assignment of y_i, we verify whether $F(y_1, \ldots, y_n, z_1, \ldots, z_m)$ evaluates to true for all assignments of $z_1, z_2, \ldots, z_m$. If all verifications hold, we accept, otherwise, we reject.

This verification process can be carried out by a polynomial-time deterministic Turing machine, making $\Sigma_2\mathbf{SAT}$ a language in Σ_2^p.

We then have to show, that every language $L \in \Sigma_2^p$ reduces to $\Sigma_2\mathbf{SAT}$. By definition, there exists a polynomial-time nondeterminitic Turing machine M such that for any string x, x is in L if and only if there exists a polynomial-size certificate y such that M accepts (x, y).

We construct a reduction from L to $\Sigma_2\mathbf{SAT}$ as follows. Given an input x, we enumerate all possible cerificates of y of polynomial size. For each certificate y, we simulate M on input (x, y). If M accepts (x, y), we construct a true totally quantified Boolean formula that encodes this acceptance, and then output the disjunction of all such formulas.

This means that, if x is in L, there exists a polynomial-size certificate y such that M accepts (x, y). the contstructed formula will have an existential quantifier for each posssible certificate, and for each existential quantifier, there exists a satisfying assignment such that M accepts. If x is not in L, then for all possible certificates y, M will not accept (x, y). Therefore, there will be no satisfying assignment for the constructed formula.

Since the reduction is polynomial-time, we have that every language $L \in \Sigma_2^p$ reduces to $\Sigma_2\mathbf{SAT}$, and thus $\Sigma_2\mathbf{SAT}$ is Σ_2^p-complete under polynomial-time reductions.

Task 7

Let's consider the original definition of **P/poly** and the proof that it equals the union of certain classes of languages decided by Turing machines with advice strings. The proof relies on a crucial property of advice strings: they are of fixed size independent of the input length. This property allows for a uniform representation of advice across all input sizes, facilitating the analysis and proof.

In the original definition of **P/poly**, advice strings are of fixed size regardless of the input length.

This uniformity ensures consistency and simplifies the analysis. Introducing advice strings that vary with input size breaks this uniformity. Each Turing machine M_n could potentially have a different advice string length $a(n)$, complicating the comparison and analysis of advice strings across different input sizes.

The proof that **P/poly** equals the union of classes of languages with fixed-size advice strings relies on careful analysis of the structure of advice strings and their relationship to the polynomial-time Turing machines. Introducing variable-length advice strings and allowing different machines for different input sizes would require a fundamentally different proof technique, as the existing proof would no longer apply.

Therefore, the proposed modification to the definition of **P/poly** raises significant challenges in maintaining the fundamental properties and consistency of the class.

Task 8

To prove $\mathbf{EXP} \not\subseteq \mathbf{SIZE}(n^k)$ for any fixed $k \in \mathbb{N}^+$, we can use a diagonalization argument. We aim to construct a language that can be decided in exponential time but cannot be computed by a circuit family of size $O(n^k)$.

We define such a language L_k, parameterized by k, as:

$$L_k = \left\{\{0,1\}^n \mid \text{the } n\text{th Turing maching halts in less than } 2^{n^k} \text{ steps}\right\}$$

Given the string $s = \{0,1\}^n$, we can simulate the nth Turing machine for 2^{n^k} steps. If it halts within the time bond, we accept, otherwise, we reject. This simulation takes at most exponential time because the number of steps is exponential in n^k.

Now we need to show that L_k cannot be decided by a circuit family of size $O(n^k)$.

We do this by contradiction. We assume that there exists such a circuit family $\{C_n\}$ where each circuit C_n has size $O(n^k)$ and decided L_k. Now consider the following construction. For each n, there are only finitely many circuits of size $O(n^k)$, so we enumarete all possible circuits of size $O(n^k)$ for input of size n. We can then arrange these circuits in some order, such that we can associate each input length n with a unique circuit. We now contruct an input x, such that it is different from the output of any circuit C_n on input of size n. We let x be the input of length n, such that the nth bit of x is the complement of the nth bit output by the nth circuit on input s. If we now consider what happens when we feed x to the circuit C_n. By construction, x differs from the output of C_n on input s at the nth bit, which means C_n cannot correctly decide whether x is in the language L_k or not. Thus we have a contradiction, and we have that there cannot exist a circuit family of size $O(n^k)$ that decides L_k.

This now proves, that L_k can be decided in exponential time, but not by a circuit family of size $O(n^k)$, and therefore, $\mathbf{EXP} \not\subseteq \mathbf{SIZE}(n^k)$ for any fixed $k \in \mathbb{N}^+$.