The Complexity Class **FP**

- Sometime, one would like to classify *functions* rather than *languages*. This can be done by slightly generalizing a couple of concepts we have previously introduced:
  - Let $T : \mathbb{N} \to \mathbb{N}$. A function f is in the class $\mathbf{FDTIME}(T(n))$ iff *there is* a TM computing f and running in time $n \mapsto c \cdot T(n)$ for some constant c.
  - The class **FP** is defined as follows, very similarly to **P**:

$$\mathbf{FP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(n^c).$$

The Complexity Class **FP**

- Sometime, one would like to classify *functions* rather than *languages*. This can be done by slightly generalizing a couple of concepts we have previously introduced:
  - Let $T : \mathbb{N} \to \mathbb{N}$. A function f is in the class $\mathbf{FDTIME}(T(n))$ iff *there is* a TM computing f and running in time $n \mapsto c \cdot T(n)$ for some constant c.
  - The class **FP** is defined as follows, very similarly to **P**:

$$\mathbf{FP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(n^c).$$

- For every $\mathcal{L} \in \mathbf{P}$, the characteristic function f of $\mathcal{L}$ is trivially in **FP**.

The Complexity Class **FP**

- Sometime, one would like to classify *functions* rather than *languages*. This can be done by slightly generalizing a couple of concepts we have previously introduced:
  - Let $T : \mathbb{N} \to \mathbb{N}$. A function f is in the class $\mathbf{FDTIME}(T(n))$ iff *there is* a TM computing f and running in time $n \mapsto c \cdot T(n)$ for some constant c.
  - The class **FP** is defined as follows, very similarly to **P**:

$$\mathbf{FP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(n^c).$$

- For every $\mathcal{L} \in \mathbf{P}$, the characteristic function f of $\mathcal{L}$ is trivially in **FP**.
- For certain classes of functions (e.g. those corresponding to optimization problems), there are canonical ways to turn a function f into a language $\mathcal{L}_f$
  - In general, however, it is not true that $f \in \mathbf{FP}$ implies $\mathcal{L}_f \in \mathbf{P}$.

Example Problems in **P** or **FP**

- **Lists**: inverting a list, sorting a list, finding the maximum or minimum element in a list, etc.

Example Problems in **P** or **FP**

- **Lists**: inverting a list, sorting a list, finding the maximum or minimum element in a list, etc.
- **Graphs**: reachability, shortest paths, minimum spanning trees, etc.

Example Problems in **P** or **FP**

- **Lists**: inverting a list, sorting a list, finding the maximum or minimum element in a list, etc.
- **Graphs**: reachability, shortest paths, minimum spanning trees, etc.
- **Numbers**: primality test, exponentiation, etc.

Example Problems in **P** or **FP**

- **Lists**: inverting a list, sorting a list, finding the maximum or minimum element in a list, etc.
- **Graphs**: reachability, shortest paths, minimum spanning trees, etc.
- **Numbers**: primality test, exponentiation, etc.
- **Strings**: string matching, approximate string matching, etc.

Example Problems in **P** or **FP**

- **Lists**: inverting a list, sorting a list, finding the maximum or minimum element in a list, etc.
- **Graphs**: reachability, shortest paths, minimum spanning trees, etc.
- **Numbers**: primality test, exponentiation, etc.
- **Strings**: string matching, approximate string matching, etc.
- **Optimization Problems**: linear programming, maximum cost flow, etc.

How to Prove a Task Being in **P** or **FP**

- In theory, one should give a TM working within some polynomial bounds, and prove that the machines decides the language (or computes the function).

How to Prove a Task Being in **P** or **FP**

- In theory, one should give a TM working within some polynomial bounds, and prove that the machines decides the language (or computes the function).
- This is however too cumbersome, and instead of going through TMs, one often goes informal and uses the so called pseudocode.
- Example.
  - Suppose you want to show the following problem to be computable in polynomial time: given two strings $x, y \in \{0,1\}^*$. determine if the x contains an instance of y.
  - A pseudocode solving the problem above is the following:

```
i ← 1;
while i ≤ |x| − |y| + 1 do
    if x[i : i + |y| − 1] = y then
        i ← i + 1
    else
        return True
    end
end
return False
```

How to Prove a Task Being in **P** or **FP**

- How could we be sure that the algorithm above indeed works in *polynomial time*?

```
i ← 1;
while i ≤ |x| − |y| + 1 do
    if x[i : i + |y| − 1] = y then
        i ← i + 1
    else
        return True
    end
end
return False
```

How to Prove a Task Being in **P** or **FP**

- How could we be sure that the algorithm above indeed works in *polynomial time*?

```
i ← 1;
while i ≤ |x| − |y| + 1 do
    if x[i : i + |y| − 1] = y then
        i ← i + 1
    else
        return True
    end
end
return False
```

- The input can be easily encoded as a binary string.

How to Prove a Task Being in **P** or **FP**

- How could we be sure that the algorithm above indeed works in *polynomial time*?

```
i ← 1;
while i ≤ |x| − |y| + 1 do
    if x[i : i + |y| − 1] = y then
        i ← i + 1
    else
        return True
    end
end
return False
```

- The input can be easily encoded as a binary string.
- The total number of instruction is polynomially bounded.
  - Indeed it is $O(|x|)$.

How to Prove a Task Being in **P** or **FP**

- How could we be sure that the algorithm above indeed works in *polynomial time*?

```
i ← 1;
while i ≤ |x| − |y| + 1 do
    if x[i : i + |y| − 1] = y then
        i ← i + 1
    else
        return True
    end
end
return False
```

- The input can be easily encoded as a binary string.
- The total number of instruction is polynomially bounded.
  - Indeed it is $O(|x|)$.
- All intermediate results are polynomially bounded in length.
  - Indeed, i cannot be greater than $O(|x|)$, thus its length is $O(\lg |x|)$.

How to Prove a Task Being in $\mathbf{P}$ or $\mathbf{FP}$

- How could we be sure that the algorithm above indeed works in *polynomial time*?

```
i ← 1;
while i ≤ |x| − |y| + 1 do
    if x[i : i + |y| − 1] = y then
        i ← i + 1
    else
        return True
    end
end
return False
```

- The input can be easily encoded as a binary string.
- The total number of instruction is polynomially bounded.
  - Indeed it is $O(|x|)$.
- All intermediate results are polynomially bounded in length.
  - Indeed, i cannot be greater than $O(|x|)$, thus its length is $O(\lg |x|)$.
- Each instruction takes polynomial time to be simulated.
  - Comparing two strings of length $|y|$ can be done in polynomial time in $|y|$, thus polynomial in $|\llcorner(x, y)\lrcorner|$.

The Class **EXP**

- What can we find if we try to go *beyond* the class **P**, i.e., if we allow TMs to have more time at their disposal?

The Class **EXP**

- What can we find if we try to go *beyond* the class **P**, i.e., if we allow TMs to have more time at their disposal?
- The *next* class of functions $T : \mathbb{N} \to \mathbb{N}$, beyond the polynomials and having nice closure properties is the class of exponential functions.

The Class **EXP**

- What can we find if we try to go *beyond* the class **P**, i.e., if we allow TMs to have more time at their disposal?
- The *next* class of functions $T : \mathbb{N} \to \mathbb{N}$, beyond the polynomials and having nice closure properties is the class of exponential functions.
- The classes **EXP** and **FEXP** are defined as follows:

$$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}(2^{n^c}) \quad \mathbf{FEXP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(2^{n^c})$$

The Class **EXP**

- What can we find if we try to go *beyond* the class **P**, i.e., if we allow TMs to have more time at their disposal?
- The *next* class of functions $T : \mathbb{N} \to \mathbb{N}$, beyond the polynomials and having nice closure properties is the class of exponential functions.
- The classes **EXP** and **FEXP** are defined as follows:

$$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}(2^{n^c}) \quad \mathbf{FEXP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(2^{n^c})$$

- The tasks in these classes *can* be solved mechanically, but *possibly cannot* be solved efficiently.

The Class $\mathbf{EXP}$

- What can we find if we try to go *beyond* the class $\mathbf{P}$, i.e., if we allow TMs to have more time at their disposal?
- The *next* class of functions $T : \mathbb{N} \to \mathbb{N}$, beyond the polynomials and having nice closure properties is the class of exponential functions.
- The classes $\mathbf{EXP}$ and $\mathbf{FEXP}$ are defined as follows:

$$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}(2^{n^c}) \qquad \mathbf{FEXP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(2^{n^c})$$

- The tasks in these classes *can* be solved mechanically, but *possibly cannot* be solved efficiently.
- Of course, it holds that

$$\mathbf{P} \subseteq \mathbf{EXP} \qquad \mathbf{FP} \subseteq \mathbf{FEXP}$$

The Class **EXP**

- What can we find if we try to go *beyond* the class **P**, i.e., if we allow TMs to have more time at their disposal?
- The *next* class of functions $T : \mathbb{N} \to \mathbb{N}$, beyond the polynomials and having nice closure properties is the class of exponential functions.
- The classes **EXP** and **FEXP** are defined as follows:

$$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}(2^{n^c}) \qquad \mathbf{FEXP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(2^{n^c})$$

- The tasks in these classes *can* be solved mechanically, but *possibly cannot* be solved efficiently.
- Of course, it holds that

$$\mathbf{P} \subseteq \mathbf{EXP} \qquad \mathbf{FP} \subseteq \mathbf{FEXP}$$

Theorem

The two inclusions above are strict.

Thank You!

Questions?