Finite Certificates

• For undecidable problems, we note that there is also a finite certificate for a ‘yes’ OR a ‘no’ answer to the problem
• Thus there is a finite certificate to show that a particular input terminates on a program.
• Similarly there is a finite certificate to prove that a word correspondence exists.
• Thus undecidable problems are know are **partially decidable**.
Two-Way Finite Certificates

• If an undecidable problem had a finite certificate for each ‘yes’ AND each ‘no’ result, we could iterate through all certificates of a problem, alternating between a ‘yes’ and a ‘no’ certificate, until a matching one is found to the input.

• Since every input has either a yes or a no certificate, this process will terminate with an answer, making the problem decidable!

• Thus no undecidable problem can have both a ‘yes’ and a ‘no’ finite certificate.
Highly undecidable

- There are those problems which are worse than the ones we talked about.
- These have no finite certificate in the ‘yes’ or ‘no’ answer.
- One example is the totality problem which asks if a program will halt on all possible inputs (halting problem asked if a program halts on a single input).
- These are worse than undecidable problems (thus highly undecidable) and there is no reduction between this and other undecidable problems.
Four Fundamental Levels of Algorithmic Behavior

- All problems fall in one of four fundamental levels of algorithmic behavior.
- Almost all problem descriptions can be modified to move from one level to another.

<table>
<thead>
<tr>
<th>Type of Problem</th>
<th>Algorithmic Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-width bounded</td>
<td>Tractable</td>
</tr>
<tr>
<td>Bounded</td>
<td>Intractable</td>
</tr>
<tr>
<td>Unbounded</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Recurring</td>
<td>Highly undecidable</td>
</tr>
</tbody>
</table>
Turing Machines and Big O

- We can easily realize that every instruction in a high-level language can be reduced to a series of finite TM transitions.
- In fact any series of instructions in a high level language (or algorithm) is polynomially reducible to a TM program.
- Thus all algorithms are polynomially equivalent to their TM counterpart and all tractable, intractable and undecidable problems remain so in TM format.
P and NP on TM

• Another definition of NP problems is that all NP problems are solved in polynomial time on a Non-Deterministic TM while P problems are those solved in polynomial time on an ordinary TM.

• Non-deterministic TM compute by running all possible paths in parallel. Yet ordinary TMs can execute non-deterministic TM programs so is NP= P?

• A normal TM can execute non-deterministic TM programs but only by trying each path sequentially, and thus would end up taking super-polynomial time. So the problem is still in NP! 