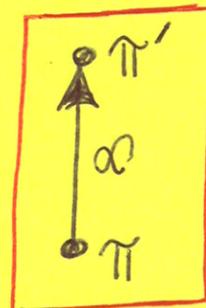
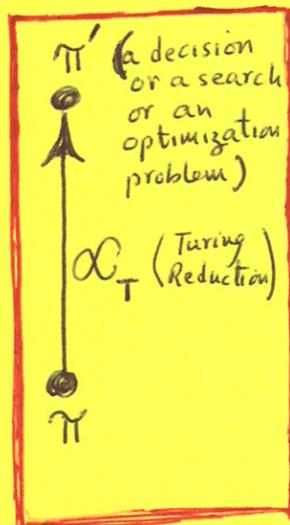


Generalizing Polynomial Transformability to Turing Reducibility and the Definition of NP-HARD

- Polynomial transformability of one problem into another made it possible for us to define the class NP-Complete of the most difficult problems in NP.
- Polynomial transformation from Π to Π' implies (among other things) that any arbitrary instance of Π can be expressed as an instance of Π' and that this mapping can be computed in polynomial time.
- But being able to construct a transformation from Π to Π' has another consequence: If we had a hypothetical algorithm that solved Π' in polynomial time, then we could solve the base problem also in polynomial time by making a subroutine call to the algorithm for Π' .
- Said another way, if the answer to the base problem Π (that's presumably difficult to solve because of its exponential complexity) can be deduced in polynomial time from the answer to Π' (regardless of whether Π' is a decision problem, or a search problem, or an optimization problem), then Π' cannot be easier to solve than Π . If Π' were to be easier to solve than Π (meaning that if Π' could be solved in polynomial time), then that would give us a way to solve Π also in polynomial time.
- "Transformation" from a base problem to a target problem (that may not necessarily be in class NP) on the basis of our ability to deduce the answer to the base problem from an imagined answer to the target problem is more aptly referred to as Turing Reducibility.



A polynomial-time Turing reduction (or, simply, Turing Reduction) from a problem Π to a problem Π' is an algorithm A that solves Π by using a hypothetical subroutine S for solving Π' such that, if S were a polynomial-time algorithm for Π' , then A would be a polynomial-time algorithm for Π .



The difference:

polynomial transformability	Turing reducibility
When we carry out a polynomial transformation from a prob. Π to a prob. Π' , for every instance $I \in D_{\Pi}$ we want $f(I) \in D_{\Pi'}$ so that I has a	There exists a Turing reduction from problem Π to problem Π' if the solution to problem Π can be deduced from the solution to problem Π' .

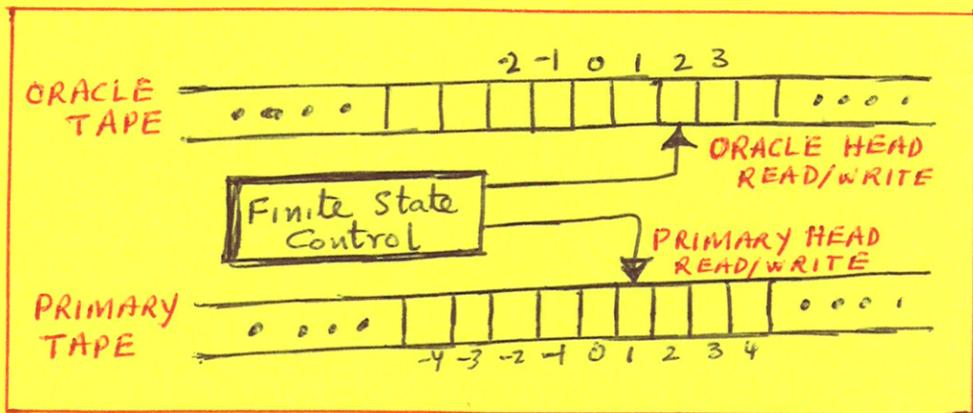
solution iff $f(I)$ has a solution.

Note that, with Turing reduction, we do NOT try to create a mapped instance $f(I) \in D_{\Pi}$, for every $I \in D_{\Pi}$. 37-2

- Turing reduction is formally defined with an Oracle Turing Machine. We will next define what we mean by an Oracle Turing Machine (OTM).

ORACLE TURING MACHINE

- An OTM consists of a standard DTM (Deterministic Turing Machine as defined in Lecture 25) augmented with an additional **oracle tape**. As with the primary tape, the squares on the oracle tape are also indexed $\dots, -2, -1, 0, 1, 2, \dots$ and this tape is scanned with a read/write oracle head



- An OTM program specifies the following:

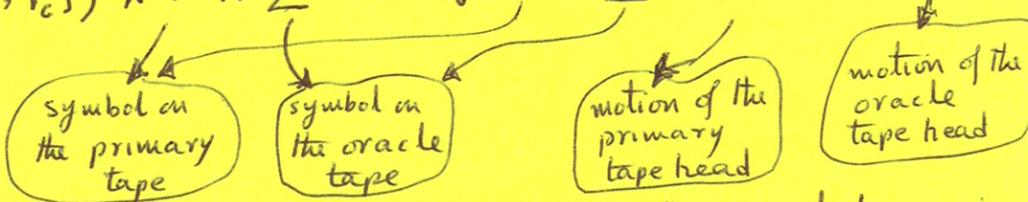
(1) A finite tape alphabet Γ which includes a subset $\Sigma \subseteq \Gamma$ as the input alphabet and a distinguished blank symbol $b \in \Gamma - \Sigma$.

(2) A finite set Q of states. Included in Q are: (i) q_0 that designates the start state; (ii) q_h that designates the halt state; (iii) q_c that designates the oracle consultation state; (iv) q_r that designates the resume computation state.

(3) A transition function:

$$\delta: (Q - \{q_h, q_c\}) \times \Gamma \times \Sigma \rightarrow Q \times \Gamma \times \Sigma \times \{-1, 1\} \times \{-1, +1\}$$

$$\delta(q, s_1, s_2) = (q', s'_1, s'_2, A_1, A_2)$$



Note that Q does NOT include states like q_y and q_N because the goal here is not to accept/reject a language. The goal here is to compute a function with the assistance of the oracle.

- The computation by an OTM on an input string $x \in \Sigma^*$ that is laid down on the primary tape is very much like that for a regular DTM, **except that when the finite-state control is in the oracle consultation state q_c , what happens next depends on a specified oracle function:**

$$g: \Sigma^* \rightarrow \Sigma^*$$

- The computation begins with the symbols of the input string x written in squares 1 through $|x|$ of the primary tape; with the rest of the primary tape and all of the oracle tape being blank; with each tape head scanning the square indexed 1 of its tape; and with the finite-state control in state q_0 .
- The computation proceeds in a step-by-step manner, with one of the following three possibilities occurring at each step:

Possibility 1 : If the current state is q_h , then the computation has ended.

Possibility 2 : If the current state is $q \in Q - \{q_c, q_h\}$, then the action taken by the finite-state control and the two read/write heads depends entirely on the transition function δ .

Possibility 3 : If the current state is the oracle consultation state q_c , then the action taken depends on the string on the oracle tape and on the oracle function g we mentioned earlier.

This is referred to as consulting the oracle

Let $y \in \Sigma^*$ be the string on the oracle tape when the finite-state control is in state q_c . And let the string $z \in \Sigma^*$ be the value of the oracle function $g(y)$. THEN IN ONE FELL SWOOP THE ORACLE TAPE'S CONTENTS ARE REPLACED BY THE STRING z IN SQUARES 1 THROUGH $|z|$. The oracle head is now set to scan the square indexed 1 and the finite-state control transitions from the state q_c to the state q_r (which is the resume-computation state).

As you would expect, the computation by an OTM program M on an input string depends on the associated oracle function g .

Let M_g denote the "relativized" OTM program obtained by combining M with the oracle g . If M_g halts for all inputs $x \in \Sigma^*$, then it can be viewed as computing a function

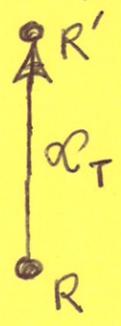
$$f_M^g : \Sigma^* \rightarrow T^*$$

defined in exactly the same way as for a DTM.

Definition of a Polynomial Time OTM Program :

M_g is a polynomial time OTM program if there exists a polynomial p such that M_g halts within $p(|x|)$ steps for every input $x \in \Sigma^*$.

Definition of Polynomial Time Turing Reduction :

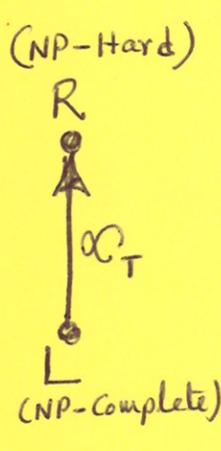


Let R and R' be any two string relations over Σ . A polynomial time Turing reduction from R to R' is an OTM program M with input alphabet Σ such that for every oracle function $g : \Sigma^* \rightarrow \Sigma^*$ that realizes R' , M_g is a polynomial time OTM program and the function f_M^g computed by M_g realizes R .

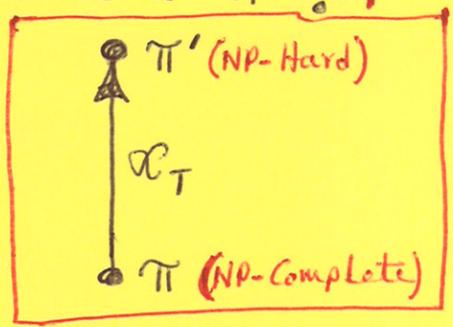
When such a reduction from R to R' exists, we write $R \leq_T R'$.

Definition of NP-Hard :

A string relation R is NP-Hard if there is some NP-Complete language L , itself stated as a string relation (in accordance with the representation of a decision problem as a search problem as explained in Lecture 36), such that $L \leq_T R$.



- A search problem Π under encoding scheme e is said to be NP-Hard if the string relation $R(\Pi, e)$ is NP-Hard.
- Informally, we say that a search problem Π' is NP-hard if there exists some NP-Complete problem Π that Turing reduces to Π' .
- Just like polynomial transformability, Turing reducibility is transitive.
- Because, as explained in Lecture 36, every decision problem can be restated as a search problem and polynomial transformability can be considered to be a special case of Turing reducibility, all NP-complete problems are NP-Hard.



As you know, a polynomial transformation from Π to Π' means calculating a mapping $f(I) \in \Pi'$ for all instances $I \in \Pi$. On the other hand, a Turing reduction from Π to Π' means that you can solve Π in polynomial time if you make a subroutine call to (oracle-supplied) the algorithm for solving Π' . Since this subroutine call could be to the mapping $f(I)$, we can think of Turing reducibility as more general than polynomial transformability.

We can therefore construct the following picture of the tractable and the intractable problems when we combine search problems with those decision problems that are in class NP :

