Polynomial transformations

a) From one language to another

Say we have two languages $L_1 \subseteq \Sigma_1^*$, and, $L_2 \subseteq \Sigma_2^*$, which are languages over the respective alphabets Σ_1 and Σ_2.

Then a polynomial transformation from L_1 to L_2 is a function f: $\Sigma_1^* \to \Sigma_2^*$ that satisfies the following two criterion:

- **1. There exists a polynomial DTM that computes f.**
- **2. For $\forall x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$.**

Notation: "L_1 transforms to L_2", $L_1 \sim L_2$, $L_1 \propto L_2$

THEOREM 1

If $L_1 \propto L_2$ then $L_2 \in$ P implies that $L_1 \in$ P and equivalently that $L_1 \notin$ P implies that $L_2 \notin$ P.

PROOF

The proof of this theorem rests on the fact that if there exists a DTM, M_1 that can recognise L_2 in polynomial time, as well as a DTM, M_2 that computes the polynomial transformation from L_1 to L_2 then we can simply combine M_1 and M_2 to make a DTM M_3 that can recognise L_1 in polynomial time.

b) From one decision problem to another

We have two decision problems Π_1 and Π_2.

Then a polynomial transformation is a function f: $\Pi_1 \rightarrow \Pi_2$ that satisfies the following two criterion:

1. **f is computable in polynomial time.**
2. **For $\forall I \in \Pi_1$, $I \in Y_{\Pi 1}$ if and only if $f(I) \in Y_{\Pi 2}$.**

THEOREM 2

Polynomial transformation is transitive:

$$L_1 \sim L_2 \wedge L_2 \sim L_3 \Rightarrow L_1 \sim L_3$$

PROOF

$f_1: \Sigma_1^* \rightarrow \Sigma_2^* \wedge f_2: \Sigma_2^* \rightarrow \Sigma_3^*$ **then**

$f(x) = f_2(f_1(x)) = f_1 \circ f_2(x): \Sigma_1^* \rightarrow \Sigma_3^*$

– **P problems form one class; they are equivalent under polynomial transformation.**

– **NP problems form another class; they are equivalent under polynomial transformation.**

NP-completeness

A language L is said to be NP-complete if $L \in$NP and for all other problems $L' \in$NP, it is the case that $L' \propto L$.

A decision problem, Π is NP-complete if the corresponding language $L[\Pi, e]$ is NP-complete for some encoding scheme e.

THEOREM

If L_1 and L_2 belong to NP, $L_1$is NP–complete, and $L_1 \sim L_2$, then L_2 is NP–complete.

PROOF

Since $L_2 \in$NP, all we need to do is show that, for every $L' \in$NP, $L' \sim L_2$. Consider any $L' \in$NP. Since L_1 is NP–complete, it must be the case that $L' \sim L_1$. The transitivity of ~ and the fact that $L_1 \sim L_2$, then imply $L' \sim L_2$.

The same will hold true for decision problems.

NP-complete problems are the hardest decision problems in the NP class. If the hardest problems in NP could be transformed in polynomial time into a problem in P, then all of the problems in NP would be in P and so then P = NP. To date no NP-complete problem has been transformed into a problem in P and therefore the majority of computer scientists believe that NP ≠ P.

Proving NP–completeness

If we just have one example of an NP-complete decision problem Π, then we can use the existence of a polynomial transformation of Π to another decision problem Π' to be a proof that Π' is also NP-complete.

To prove that Π' is NP–complete show that:

1. **$\Pi' \in$ NP, and**
2. **some known NP–complete problem Π transforms to Π'.**

The problem of satisfiability: SAT

Satisfiability of first order logical clauses in conjunctive normal form CNF.

$U = \{u_1, u_2, \ldots, u_n\}$ **set of *Boolean variables*.**

Truth assignment function $t: U \rightarrow \{T, F\}$.

For each $u \in U$ we say that both u and its negation $\bar{u}$ are literals over the set of variables U. $\bar{u}$ is defined such that if $t(u) = T$ then $t(\bar{u}) = F$, otherwise $t(\bar{u}) = T$.

A *clause* over U is a set of literals over U, such as $\{u_1, \bar{u}_3, u_8\}$, each of which consists of the disjunction of some set of literals over U, and *satisfied* by a truth assignment if and only if at least one of its member is true under that truth assignment.

A collection C of clauses over U is *satisfiable* iff there exists some truth assignment for U that simultaneously satisfies all the clauses in C. → *satisfying truth assignment*

The Variables

- **Q[i,qk] where i runs from 0 to p(n) and qk runs through all states of M**
- **H[i,j] where i runs from 0 to p(n) and j runs from -p(n) through p(n)+1**
- **S[i,j,sk] where i runs from 0 to p(n), j runs from -p(n) through p(n)+1, and sk runs through all symbols of T (tape symbols)**

The Meaning of the Variables

- **Q[i,qj] means that at time i, M is in state qj**
- **H[i,j] means that at time i, M is scanning tape square j. Note that in p(n) transitions, the read-write head can move at most distance p(n) from its starting point.**
- **S[i,j,sk] means that at time i, the contents of tape square j is sk.**

Clause Groups

- **G1 - Guarantee that at each time i, M is in one and only one state**
- **G2 - Guarantee that at each time i, M is scanning one and only one tape square**
- **G3 - Guarantee that at each time i, there is one and only one symbol in each tape square of the used tape**
- **G4 - Guarantee that the machine starts in q0 with x properly positioned on the tape and the read-write head properly positioned.**
- **G5 - Guarantee that by time p(n) M has entered qy.**
- **G6 - Guarantee that the transitions are applied properly**

Group G1

- For each time i, add the clause {Q[i,q1],Q[i,q2], … , Q[i,qt]} where t is the number of states in Q.
- For each time i, add the set of clauses $\{\overline{Q[i,qk]},\overline{Q[i,qj]}\}$ where k and j, taken together run through all pairs of states of Q. If Q has t states then t(t+1)/2 clauses are required for each time i.

The first part guarantees that at each time i, M is in at least one state. The second part (with the paired barred variables) guarantees that M is not in more than one state at time i. The time i runs from 0 through p(n).

Group G2

- For each time i, add the clause: {H[i,-p(n)],H[i,-p(n)+1],…,H[i,p(n)+1]}
- For each time i, let j and k run through all possible pairs of tape squares from -p(n) to p(n)+1. For each pair (j,k), and each time i, add the clause $\{\overline{H[i,j]},\overline{H[i,k]}\}$.

The first clause says that M must be scanning at least one tape square at every time i. The second set of clauses says that M cannot be scanning more than one tape square at any given time i.

Group G3

- Let i run through all times from 0 to p(n) and j run through all tape squares from -p(n) through p(n)+1. (There are p(n)*2(p(n)+1) combinations.
- For each (i,j) add {S[i,j,s0],S[i,j,s1], … ,S[i,j,sk]}, where s0,s1, … ,sk run through all tape symbols in T.
- Let l and m run through all pairs of tape symbols. If there are k tape symbols, then there are k(k+1)/2 pairs.
- For each combination (i,j) and each pair (l,m), add the following clause $\{\overline{S[i,j,l]},\overline{S[i,j,m]}\}$

G3 Clauses model the behavior of the tape. The first set of clauses guarantees that at any time i, each tape square contains at least one tape symbol. We are concerned only about squares numbered from -p(n) through p(n)+1. The second set of clauses guarantees that at any time i, no tape square contains more than one tape symbol.

Group G4

Add {Q[0,q0]} :we start in state 0.
Add {H[0,1]} : the read-write head starts with square 1.
Add {S[0,1,x1]}, {S[0,1,x2]}, … ,{S[0,n,xn]} : the input string is on the tape in the correct position at time 0.
Add {S[0,0,b]}
Add {S[0,n+1,b]}, {S[0,n+2,b]}, … , {S[0,p(n)+1,b]}
Add {S[0,-1,b]}, {S[0,-2,b]}, … , {S[0,-p(n),b]}

The final sets of clauses guarantee that at time 0, the rest of the tape is blank.

Group G5

- Add {Q[p(n),qy]}

Once we enter state qy, no further transitions are allowed. This clause guarantees that we have entered state qy either at some time prior to p(n) or at time p(n). Entering qy causes M to accept its input.

Group G6

- Let (qa,sb,qc,sd,e) be an element of δ, where e is an element of {L,R}.
- We need to model the following logical statement in CNF form: If the current time is i and M is in state qa and X is scanning tape square j and tape square j contains symbol sb, then at time i+1, MX will be in state qb, tape square j will contain sd and MX will be scanning either square j+1 or j-1 depending on e.
- If P then Q is logically equivalent to ~P OR Q.
- Assume e=L, then using the variables we get: ~(Q[i,qa] AND H[i,j] AND S[i,j,sb]) OR (Q[i+1,qb] AND H[i+1,j+1] AND S[i+1,j,sd])
- For e=R, ~(Q[i,qa] AND H[i,j] AND S[i,j,sb]) OR (Q[i+1,qb] AND H[i+1,j-1] AND S[i+1,j,sd])

Deriving CNF Form

- ~(Q[i,qa] AND H[i,j] AND S[i,j,sb]) OR (Q[i+1,qb] AND H[i+1,j+1] AND S[i+1,j,sd])
- DeMorgan's Law: ($\overline{Q[i,qa]}$ OR $\overline{H[i,j]}$ OR $\overline{S[i,j,sb]}$) OR (Q[i+1,qb] AND H[i+1,j+1] AND S[i+1,j,sd])
- Apply Distributive Law to obtain Three Clauses

Final Group

e=L

{$\overline{Q[i,qa]}$, $\overline{H[i,j]}$, $\overline{S[i,j,sb]}$, Q[i+1,qb]}
{$\overline{Q[i,qa]}$, $\overline{H[i,j]}$, $\overline{S[i,j,sb]}$, H[i+1,j+1]}
{$\overline{Q[i,qa]}$, $\overline{H[i,j]}$, $\overline{S[i,j,sb]}$, S[i+1,j,sd]}

e=R

{$\overline{Q[i,qa]}$, $\overline{H[i,j]}$, $\overline{S[i,j,sb]}$, Q[i+1,qb]}
{$\overline{Q[i,qa]}$, $\overline{H[i,j]}$, $\overline{S[i,j,sb]}$, H[i+1,j-1]}
{$\overline{Q[i,qa]}$, $\overline{H[i,j]}$, $\overline{S[i,j,sb]}$, S[i+1,j,sd]}

- For each element of δ, add one three-clause group for each combination of time i, and tape square j.
- For each element of δ, we generate 3*p(n)*2(p(n)+1) clauses.

The final Boolean Expression is E=G1∪G2∪G3∪G4∪G5∪G6