Chapter 15

P, *NP*, and Cook's Theorem

Computability Theory

- Establishes whether decision problems are (only) theoretically decidable, i.e., decides whether each solvable problem has a practical solution that can be solved *efficiently*
- A theoretically solvable problem may not have a practical solution, i.e., there is <u>no</u> efficient algorithm to solve the problem in *polynomial* time – an *intractable problem*
  - Solving intractable problems require extraordinary amount of time and memory.
  - *Efficiently solvable* problems are *polynomial* (*P*) problems.
  - *Intractable* problems are *non-polynomial* (*NP*) problems.
- Can any problem that is solvable in polynomial time by a non-deterministic algorithm also be solved deterministically in polynomial time, i.e., *P* = *NP* ?

15.1 Time Complexity of NTMs

- A deterministic TM searches for a solution to a problem by sequentially examining a number of possibilities, e.g., to determine a perfect square number.

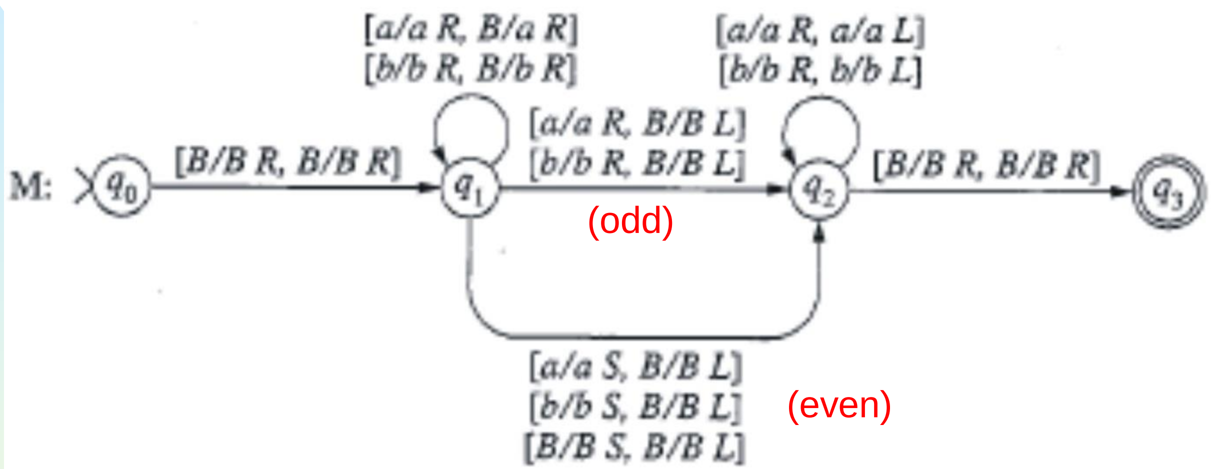
- A NTM employs a "guess-and-check" strategy on any one of the possibilities.

- Defn. 15.1.1 The time complexity of a NTM *M* is the function tc_M: $\mathbf{N} \to \mathbf{N}$ such that $tc_M(n)$ is the *maximum* number of transitions in any computation for an input of length *n*.

- Time complexity measures the *efficiency* over all computations
  - the *non-deterministic* analysis must consider **all** possible computations for an input string.
  - the guess-and-check strategy is generally *simpler* than its deterministic counterparts.

15.1 Time Complexity of NTMs

- <u>Example 15.1.1</u> Consider the following two-tape NTM *M* that accepts the palindromes over {*a*, *b*}.



  - the time complexity of *M* is

$$tc_M(n) = \begin{cases} n+2 & \text{if } n \text{ is odd} \\ n+3 & \text{if } n \text{ is even} \end{cases}$$

- The strategy employed in the transformation of a NTM to an equivalent DTM (given in Section 8.7) does <u>not</u> preserve *polynomial time solvability*.

15.1 Time Complexity of NTMs

- Theorem 15.1.2 Let L be the language accepted by a one-tape NTM M with time complexity $tc_M(n) = f(n)$. Then L is accepted by a DTM M' with time complexity $tc_{M'}(n) \in O(f(n)c^{f(n)})$, where c is the maximum number of transitions for any <state, symbol> pair of M.

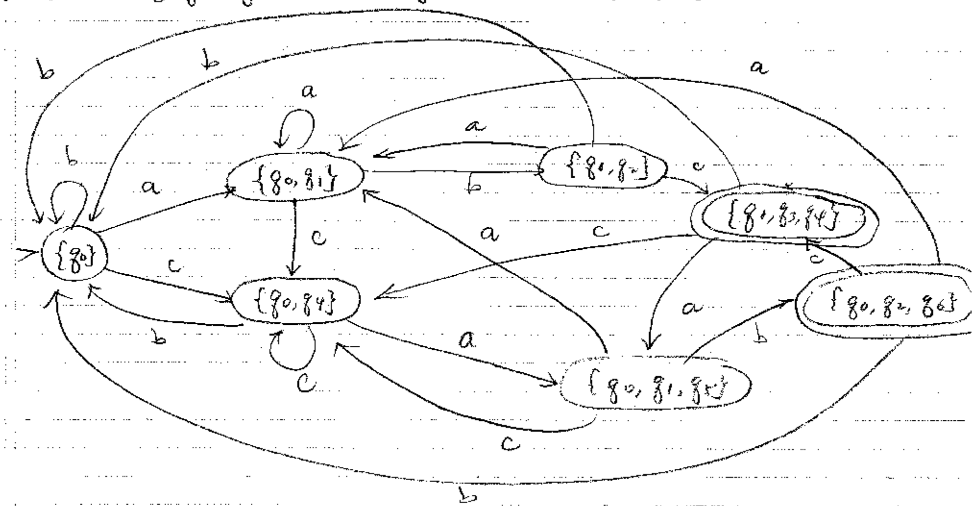
  - Proof. Let M be a one-tape NTM that halts for all inputs, and let c be the maximum number of distinct transitions for any <state, symbol> pair of M. The transformation from non-determinism to determinism is obtained by associating a unique computation of M with a sequence $(m_1, \ldots, m_n)$, where $1 \leq m_i \leq c$. The value m_i indicates which of the c possible transitions of M should be executed on the i^{th} step of the computation.

    A three-tape DTM M' was described in Section 8.7 (pages 275-277) whose computation with input w iteratively simulated all possible computations of M with input w.

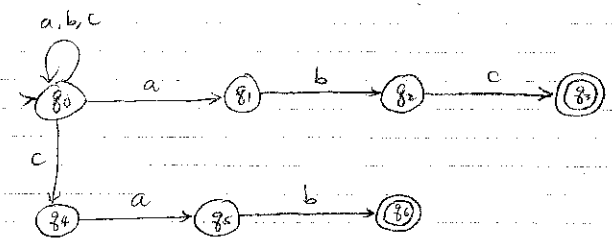
Theorem 15.1.2 (Continued)

DTM:

t	a	b	c
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_4\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_4\}$
$\{q_0, q_4\}$	$\{q_0, q_1, q_5\}$	$\{q_0\}$	$\{q_0, q_4\}$
$\{q_0, q_1, q_5\}$	$\{q_0, q_1\}$	$\{q_0, q_2, q_6\}$	$\{q_0, q_4\}$
$* \{q_0, q_2, q_6\}$	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_3, q_4\}$
$* \{q_0, q_3, q_4\}$	$\{q_0, q_1, q_5\}$	$\{q_0\}$	$\{q_0, q_4\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_3, q_4\}$



Original NTM:



15.1 Time Complexity of NTMs

- Theorem 15.1.2 (Cont.) Given a NTM *M* with $tc_M(n) = f(n)$, show a DTM *M*' with time complexity $tc_{M'}(n) \in O(f(n)c^{f(n)})$, where *c* = max. no. of transitions for any <state, symbol> pair of *M*.
  - Proof. (Cont.) We analyze the number of transitions required by *M*' to simulate all computations of *M*.

    For an input of length *n*, the max. no. of transitions in *M* is at most $f(n)$. To simulate a single computation of *M*, *M*' behaves as follows:

    1) generates a sequence $(m_1, \ldots, m_n)$ of transitions, $1 \leq m_i \leq c$
    2) simulates the computation of *M* using $(m_1, \ldots, m_n)$, and
    3) if the computation does not accept the input string, the computation of *M*' continues with Step 1.

    In the worst case, $c^{f(n)}$ sequences are examined for each single computation of *M*.

    As the simulation of a computation of *M* can be performed using $O(f(n))$ transitions of *M*', $tc_{M'}(n) \in O(f(n)c^{f(n)})$ in simulating *M* by *M*'.

15.2 The Classes *P* and *NP*

- Defn. 15.2.1 A language L is decidable in polynomial time if there is a standard TM M that accepts L with $tc_M \in O(n^r)$. The family of languages decidable in polynomial time is denotes P.

- Any problem that is polynomially solvable on a standard TM is in P, and the choice of DTM models (e.g., multi-tape, multi-track) for the analysis is invariant.

- Defn. 15.2.2 A language L is accepted in nondeterministic polynomial time if there is a NTM M that accepts L with $tc_M \in O(n^r)$. The family of languages accepted in non-deterministic polynomial time is denoted NP.

- Since every DTM is a NTM, $P \subseteq NP$.

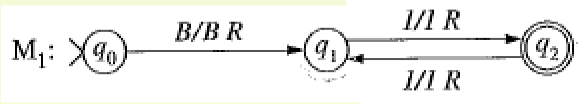
- The family NP is a subset *of* the *recursive languages*, since the number of transitions ensure all computations terminates

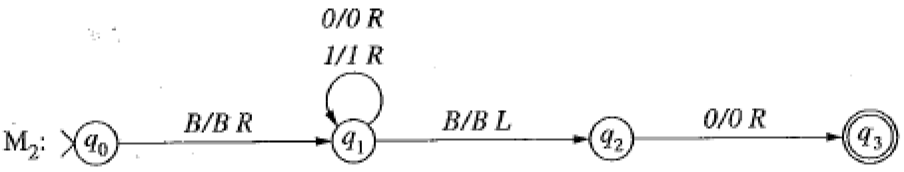
15.3 Problem Representation and Complexity

- Design a TM *M* to solve a decision problem *R* consists of 2 steps:
  1. Represent the instances of *R* as strings
  2. Construct *M* that analyzes the strings and solves *R*
  - which requires the discovery of an algorithm to solve *R*

- The time complexity (*tc*) of a TM relates the *length* of the input to the *number of transitions* in the computations, and thus the selection of the representation have direct impacts on the computations.

- Example. Given the following TMs M_1 (encodes *n* as 1^{n+1}) and M_2 (encodes *n* by the standard binary representation):





  - where M_1 and M_2 both solve the problem of deciding whether a natural number is even, with the inputs to M_1 using the *unary* representation and M_2 the *binary* representation.

15.3 Problem Representation and Complexity

- Example. (Cont.)
  - The $tc_{M_1} = tc_{M_2} \in O(n)$ and the difference in representation does not affect the complexity; however, the modification (shown below) has a significant impact on the complexity.
  - Consider TM M_3, which includes a TM T that transforms an input in *binary* to its *unary* in solving the same problem:

M_3: Binary representation → [T] → Unary representation → [M_1] → Yes / → No

  - The *complexity* of the new solution, i.e., M_3, is analyzed in the following table, which shows the *increase* in string length caused by the conversion:

String Length	Maximum Binary Number	Decimal Value	Unary Representation
1	1	1	$11 = 1^2$
2	11	3	$1111 = 1^4$
3	111	7	$11111111 = 1^8$
i	1^i	$2^i - 1$	1^{2^i}

15.3 Problem Representation and Complexity

- <u>Example</u>. (Cont.)

(Binary) String Length	Max. Binary No.	Decimal Value	Unary Representation
1	1	1	$11 = 1^2$
2	11	3	$1111 = 1^4$
3	111	7	$11111111 = 1^8$
i	1^i	$2^i - 1$	1^{2^i}

  - tc_{M_3} is determined by the complexity of T and M_1.
  - For the input of length i, the string 1^i requires the maximum number of transitions of M_3, i.e.,

$$tc_{M_3}(n) = tc_T(n) + tc_{M_1}(2^n)$$

$$= tc_T(n) + 2^n + 1$$

which is *exponential* even without adding tc_T. The *increase* in the complexity is caused by the *increase* in the *length* of the input string using the <u>unary</u> representation.

15.4 Decision Problems & Complexity Classes

- Decision problems from *P* and *NP*

Acceptance of Palindromes
Input: String *u* over alphabet Σ
Output: *yes* – *u* is a palindrome
no – otherwise
Complexity – in *P* (O(n^2), p. 444)

Derivability in CNF Grammar
Input: CNF grammar *G*, string *w*
Output: *yes* – if $S \stackrel{*}{\Rightarrow} w$
no – otherwise
Complexity – in *P* (CYK Alg: O(n^3), p. 124)

Subset Sum Problem
Input: Set *S*, $v: S \to \mathbf{N}$, *k*
Output: *yes* – if $\exists$ S' ($\subseteq$ S) whose total value is *k*
no – otherwise
Complexity – in *P* (unknown)
– in *NP* (Yes)

Path Problem for Directed Graphs
Input: Graph $G = (N, A)$, $v_i, v_j \in N$
Output: *yes* – if $\exists$ path(v_i, v_j) in *G*
no – otherwise
Complexity – in *P* (Dijkstra's alg: O(n^2))

Hamiltonian Circuit Problem
Input: Directed graph $G = (N, A)$
Output: *yes* – if $\exists$ cycle with each vertex in G
no – otherwise
Complexity – in *P* (unknown)
– in *NP* (Yes)

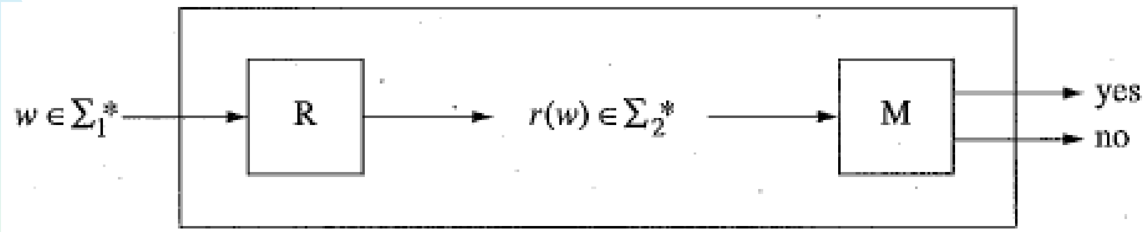
- Each of the *NP* problems can be solved *non-deterministically* using a "guess-and-check" strategy

15.6 Polynomial-Time Reduction

- *Reduction* is a problem-solving technique employed to
  - avoid "reinventing the wheel" when encountering a new problem
  - transform the instances of the new problem into those of a problem that has been solved
  - establish the *decidability* and *tractability* of problems
- Defn. 11.3.1 Let L be a language over alphabet Σ_1 and Q be a language over Σ_2. L is many-to-one reducible to Q if there exists a *Turing computable function* $r : \Sigma_1^* \rightarrow \Sigma_2^*$ such that $w \in L$ if, and only if, $r(w) \in Q$.
  - if a language L is reducible to a *decidable* language Q by a function r, then L is also *decidable*.

15.6 Polynomial-Time Reduction

- Example (p. 348). Let *R* be the TM that computes the *reduction*, i.e., input(*L*) to input(*Q*), and *M* the TM that accepts language *Q*. The sequential execution of *R* and *M* on strings from Σ_1^* accepts language *L* (by accepting inputs to Q) is



  - *R*, the reduction TM, which does not determine membership in either *L* or *Q*, transforms strings from Σ_1^* to Σ_2^*.
  - Strings in *Q* are determined by *M*, and strings in *L* are by the combination of *R* and *M*.

15.6 Polynomial-Time Reduction

- A reduction of a language L to a language Q transforms the question of membership in L to that of membership in Q.
  - Let r be a reduction (function) of L to Q computed by a TM R. If Q is accepted by a TM M, then L is accepted by a TM that

    i) runs R on input string $w \in \Sigma_1^*$, and

    ii) runs M on $r(w)$.

    The string $r(w)$ is accepted by M if, and only if, $w \in L$

  - The time complexity includes

    i) time required to transform the instances of L, and

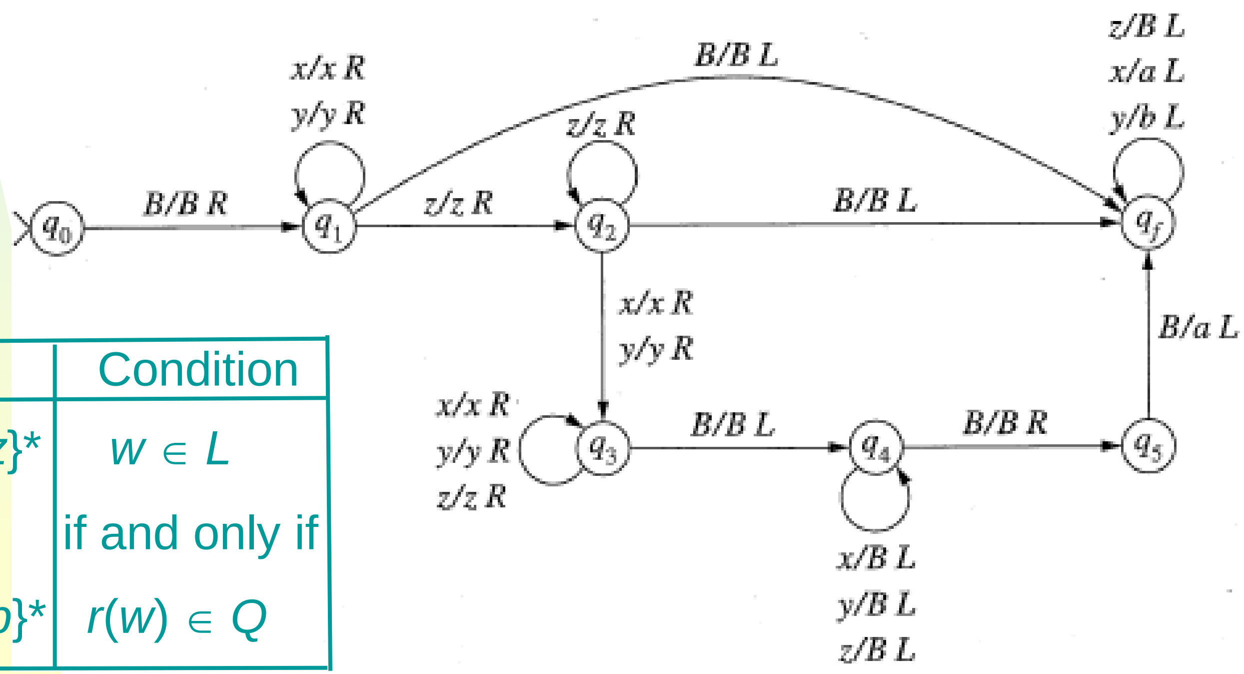
    ii) time required by the solution to Q.

- Defn. 15.6.1 Let L and Q be languages over alphabets Σ_1 and Σ_2, respectively. L is reducible in polynomial time to Q if there is a polynomial-time computable function $r : \Sigma_1 \to \Sigma_2$ such that $w \in L$ if, and only if, $r(w) \in Q$.

15.6 Polynomial-Time Reduction

- Example 15.6.1 (p. 349, 478) Reduces $L = \{ x^i y^i z^k \mid i \geq 0, k \geq 0 \}$ to $Q = \{a^i b^i \mid i \geq 0\}$ by transforming $w \in \{x, y, z\}^*$ to $r(w) \in \{a, b\}^*$.
  - If w $\in x^* y^* z^*$, replace each '*x*' by '*a*' and '*y*' by '*b*', and erase the *z*'s
  - otherwise, replace *w* by a single '*a*'

The following TM transforms multiple strings in *L* to the same string in *Q* (i.e., a many-to-one reduction):



Reduction	Input	Condition
L	$w \in \{x, y, z\}^*$	$w \in L$
↓	↓ *r*	if and only if
Q	$r(w) \in \{a, b\}^*$	$r(w) \in Q$

15.6 Polynomial-Time Reduction

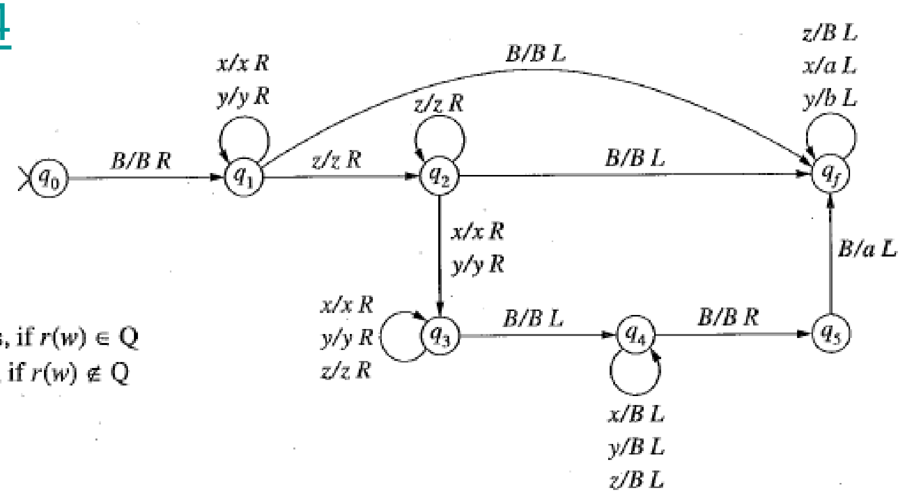
- <u>Theorem 15.6.2</u> Let L be reducible to Q in *polynomial time* and let $Q \in$ P. Then $L \in$ P.
  - <u>Proof</u>. Let R denote the TM that computes the reduction of L to Q and M the TM that decides Q. L is accepted by a TM that sequentially run R and M. The time complexities tc_R and tc_M combine to produce an *upper bound* on the no. of transitions of a computation of the composite TM. The computation of R with input string w generates the string $r(w)$, which is the input to M. The function tc_R can be used to establish a bound on the length of $r(w)$. If the input string w to R has length n, then the length of $r(w)$ cannot exceed the max(n, $tc_R(n)$).

    A computation of M processes at most $tc_M(k)$ transitions, where k is the length of its input string. The number of transitions of the composite TM (i.e., R and M) is bounded by the sum of the estimates of R and M. If $tc_R \in O(n^s)$ and $tc_M \in O(n^t)$, then

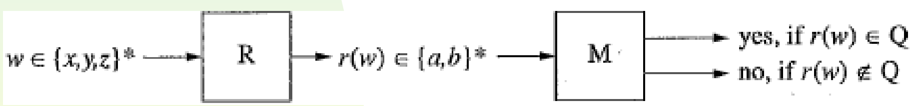
$$tc_R(n) + tc_M(tc_R(n)) \in O(n^{st})$$

15.6 Polynomial-Time Reduction

- <u>Example 15.6.1</u> (Continued) Reduces $L = \{ x^i y^i z^k \mid i \geq 0,\ k \geq 0 \}$ to $Q = \{ a^i b^i \mid i \geq 0 \}$:
  - For string n of length ≥ 0, $tc_R(0) = 2$, $tc_R(1) = 4$, $tc_R(2) = 8$, etc.
  - The worst case occurs for the remainder of the strings when an 'x' or 'y' follows a 'z', i.e., when w is read in q_1, q_2, and q_3, and erased in q_4. The computation is completed by setting $r(w) = a$, and for $n > 1$, <u>$tc_R(n) = 2n + 4$</u>



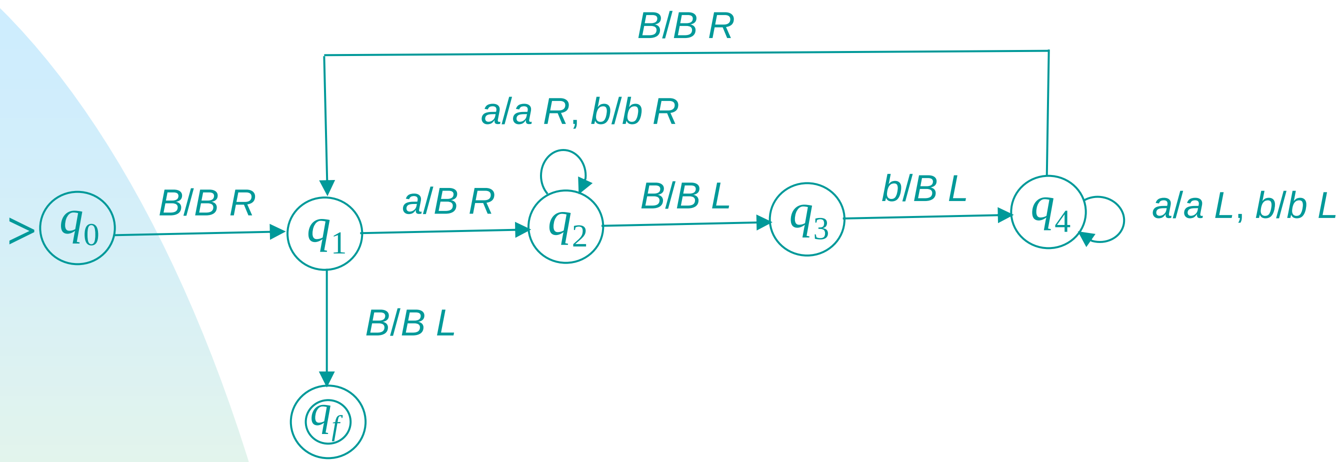
  - Combining R and M

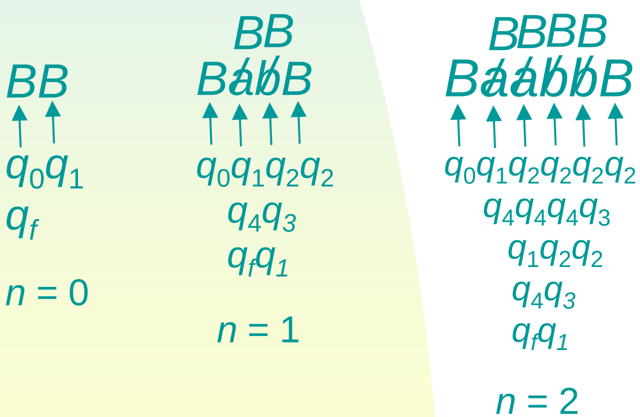


  - The combined TM accepts Q with $tc_M(n) =$ <u>$2n^2 + 3n + 2$</u>.
  - Worst-case(tc_M): input $a^{n/2}b^{n/2}$, if n is even, or $a^{(n-1)/2}b^{(n-1)/2}$, if n is odd
  - Thus, $tc_R(n) + tc_M(tc_R(n)) = (2n + 4) + (2(2n+4)^2 + 3(2n + 4) + 2) \in O(n^2)$. The upper bound in Theorem 15.6.2, i.e., $tc_R(n) + tc_M(tc_R(n)) \in O(n^{st})$.

15.6 Polynomial-Time Reduction

- Example. A TM M that accepts $Q = \{ a^n b^n \mid n \geq 0 \}$ and its tc:





n	$tc_M(n)$
0	2
1	7
2	16
3	29
4	46
:	:

Iteration	Move	Steps
1	R	$2n+1$
	L	$2n$
2	R	$2n-1$
	L	$2n-2$
:	:	:

$$tc_M(n) = 2n^2 + 3n + 2 \in O(n^2)$$

15.7 *P* = *NP*?

- A language accepted in *polynomial time* by DTM with multi-track or -tape is in *P*.

- The process for constructing an equivalent DTM from a NTM does <u>not</u> preserve polynomial-time complexity. (See Theorem 15.1.2: $tc_M(n) = f(n) \Rightarrow tc_{M'}(n) \in O(f(n)c^{f(n)})$.)

- Due to the additional time complexity of currently known non-deterministic solutions over deterministic solutions across a wide range of important problems, it is generally believe that $P \neq NP$.

- The $P = (\neq)$ *NP* problem is a precisely formulated mathematical problem and will be resolved only when either (i) the *equality* of the two classes, or (ii) $P \subset NP$ is proved.

- <u>Defn. 15.7.1</u> A language *Q* is called NP-hard if for every $L \in$ NP, *L* is reducible to *Q* in polynomial time. An NP-hard language that is also in NP is called NP-complete.

15.7 $P = NP$?

- Some problems L are so hard that although we can prove they are NP-hard, we cannot prove they are NP-complete, i.e., $L \in NP$.

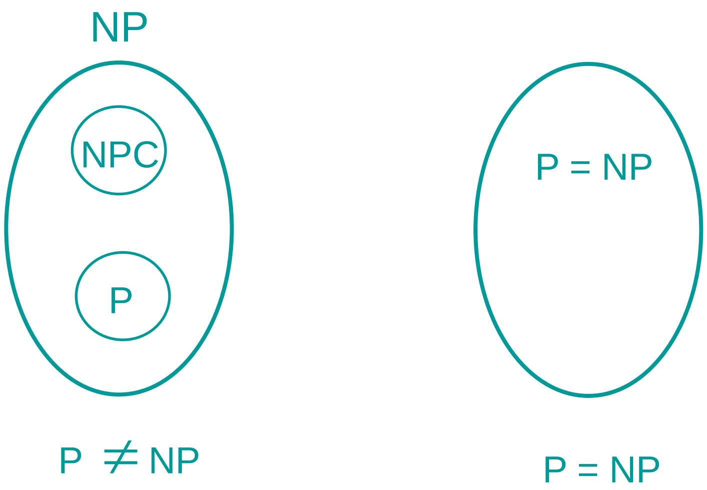
- $P = NP$, if there exists a polynomial-time TM, which accepts an NP-complete language, can be used to construct TMs to accept every language in NP in deterministic polynomial time.

- Theorem 15.7.2 If there is an NP-hard language that is also in P, then $P = NP$.

  - Proof. Assume that Q is an NP-hard language that is accepted in polynomial time by a DTM, i.e., $Q \in P$. Let $L \in NP$. Since (by Defn. 15.7.1) Q is NP-hard, there is a polynomial time reduction of L to Q. By Theorem 15.6.2 (which states that if L is reducible to Q in polynomial time and $Q \in P$, then $L \in P$), $L \in P$.

15.9 Complexity Class Relations

- The class consisting of all NP-complete problems, which is non-empty, is denoted NPC.
  - If $P \neq NP$, then P and NPC are nonempty, disjoint subsets of NP, which is the scenario believed to be true by most mathematicians and computer scientists.
  - If $P = NP$, then the two sets collapse to a single class.



15.8 The Satisfiability Problem

- The Satisfiability Problem
  - An NP-complete problem
  - Determines whether there is an assignment of truth values to propositions that makes a formula true
  - The truth value of a *formula* is obtained from those of the elementary *propositions* occurring in the formula
- Fundamentals of Propositional Logic
  - A Boolean variable, which takes on the values 0 & 1, is considered to be a *proposition*
  - The value of a variable specifies the truth/falsity of the proposition
  - The logical connectives $\wedge$ (and), $\vee$ (or), and $\neg$ (not) are used to construct propositions, i.e., well-formed formulas (wff), from a set of Boolean variables

15.8 The Satisfiability Problem

- Propositional Logic
  - A clause is a well-formed formula that consists of a <u>disjunction</u> of variables or the <u>negation</u> of variables in which an *unnegated* (*negated*) *variable* is called a *positive* (*negative*) literal
  - A formula is in conjunctive normal form (CNF) if it has the form $u_1 \wedge u_2 \wedge u_n$, where each u_i $(1 \leq i \leq n)$ is a clause, e.g.,

$$(x \vee \neg y \vee \neg z) \wedge (x \vee z) \wedge (\neg x \vee \neg y)$$

- The Satisfiability Problem is the problem of deciding if a CNF is satisfied by some truth assignment, e.g., the above CNF is satisfied by $x = 1$, $y = 0$, and $z = 0$
- A deterministic solution to the Satisfiability Problem can be obtained by checking every truth assignment, in which the number of possible truth assignments is 2^n, where n is the number of *Boolean variables*

15.8 The Satisfiability Problem

- Theorem 15.8.2 The Satisfiability Problem is in *NP*

  Proof. A representation of the wff over a set of Boolean variables $\{x_1, x_2, ..., x_n\}$ such that (i) a variable is encoded by the binary representation of its subscript, and (ii) a literal L is the encoding of its variable followed by #1 if L is *positive*, and 0, otherwise. For example,

  $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$ is encoded as $1\#1 \vee 10\#0 \wedge 1\#0 \vee 11\#1$

  An input to TM M consists of the encoding of the *variables* in the wff followed by ## & the encoding of the wff, e.g.,

  $1 \# 10 \# 11 \#\# 1\#1 \vee 10\#0 \wedge 1\#0 \vee 11\#1$

  The language L_{SAT} consists of all string over $\Sigma = \{ 0, 1, \wedge, \vee, \# \}$ that represent satisfiable CNF formula.

  A two-tape NTM M that solves the Satisfiability Problem non-deterministically generates a *truth assignment*. The initial setup contains the representation of the wff on tape 1 w/ tape 2 blank.

15.8 The Satisfiability Problem

e.g., Tape 2 BB

Tape 1 $B1\#10\#11\#\#1\#1 \vee 10\#0 \wedge 1\#0 \vee 11\#1B$

1. If the input does not have the anticipated form, the computation halts and rejects the string.

2. The encoding of x_1 on tape 1 is copied onto tape 2, which is followed by printing # and non-deterministically writing 0 or 1, encoded as $t(x_1)$, i.e., the truth assignment of x_1.

If this is not the last variable, ## is written and the step is repeated for the next variable. For example,

Tape 2 $B1\#t(x_1)\#\#10\#t(x_2)\#\#11\#t(x_3)B$

Tape 1 $B1\#10\#11\#\#1\#1 \vee 10\#0 \wedge 1\#0 \vee 11\#1B$

The tape head on tape 2 is repositioned at the leftmost position. The head on tape 1 is moved past ## into a position to read the 1st variable of the wff.

15.8 The Satisfiability Problem

3. Assume that the encoding of the variable x_i is scanned on tape 1. The encoding of x_i is found on tape 2. M compares the value $t(x_i)$ on tape 2 with the Boolean value following x_i on tape 1.

4. If the values do not match, the current literal is not satisfied by the truth assignment.

   If the symbol following the literal is a B or $\wedge$, every literal in the current clause has been examined & failed. When this occurs; the truth assignment does not satisfy the wff & the computation halts in a non-accepting state.

   If $\vee$ is read instead, the tape heads are positioned to examine the next literal in the clause (step 3).

5. If the values do match, the literal & current clause are satisfied by the truth assignment. The head on tape 1 moves to the right to the next $\wedge$ or B.

   If a B is found, the computation halts & accepts the input. Otherwise, the next clause is processed by returning to step 3.

15.8 The Satisfiability Problem

- The matching procedure in step 3 determines the rate of growth of the time complexity of M.

In the worst case, the matching requires comparing each variable on tape 1 with each of the variables on tape 2 to discover the match. This can be accomplished in $O(k \times n^2)$ time, where

- n is the number of variables, and
- k is the number of literals in the input

15.8 The Satisfiability Problem

- Theorem 15.8.3 The Satisfiability Problem is NP-hard.

  Proof. Let L be a language accepted by a NTM M whose computations are bounded by a polynomial *p*. The reduction of L to the Satisfiability Problem is achieved by transforming the computations of M with an input string *u* into a CNF formula *f*(*u*) so that $u \in L(M)$ iff *f*(*u*) is *satisfiable*. The construction of *f*(*u*) is then shown to require time that grows only polynomially w/ |*u*|.

  It is assumed that all computations of M halt in one of 2 states, the *accepting* state q_A and *rejecting* state q_R. It is assumed that there are no transitions leaving these states.

  An arbitrary TM can be transformed into M satisfying these restrictions by adding transitions from every accepting configuration to q_A and from every rejecting configuration to q_R . The transformation from a computation to a wff assumes that all computations with input of length *n* contain *p*(*n*) configurations.

15.8 The Satisfiability Problem

- Proof (Continued). The (final) states and alphabets of M are denoted

  $Q = \{ q_0, q_1, \ldots, q_m \}$
  $\Gamma = \{ B, a_0, a_1, \ldots, a_s, a_{s+1}, \ldots, a_t \}$
  $\Sigma = \{ a_{s+1}, a_{s+2}, \ldots, a_t \}$
  $F = \{ q_m \}$, and q_{m-1} is the lone *rejecting* state

  Let $u \in \Sigma^*$ be a string of length n. A wff $f(u)$ is defined that encodes the computations of M with input u. The length of $f(u)$ depends on $p(n)$, the max. no. of computation of M with input of |n|.

  The encoding is designed so that there is a *truth assignment* satisfying $f(u)$ iff $u \in L(M)$. The wff is built from three classes of variables which represent a property of a machine configuration.

Variable		**Interpretation (when satisfied)**
$Q_{i,k}$	$0 \leq i \leq m$, $0 \leq k \leq p(n)$	M is in *state* q_i at *time* (transition) k
$P_{j,k}$	$0 \leq j \leq p(n)$, $0 \leq k \leq p(n)$	M scans *position* j at *time* k
$S_{j,r,k}$	$0 \leq j \leq p(n)$, $0 \leq r \leq t$, $0 \leq k \leq p(n)$	Tape position j contains symbol a_r at time k

15.8 The Satisfiability Problem

- Proof (Continued). The set of variables *V* in a wff is the *union* of the three sets defined above. A computation of M defines a truth assignment on *V*. For example, if tape position 3 initially contains symbol a_i, then $S_{3,i,0}$ is *true* and $S_{3,j,0}$ must be *false*, $\forall_{i \neq j}$.

  A truth assignment obtained in this manner specifies (i) the *state*, (ii) *position* of the tape head, and (iii) the *symbols* on the tape for each time *k* ($0 \leq k \leq p(n)$). This is the information contained in the sequence of configurations produced by the computation.

  An arbitrary assignment of truth values to the variables in *V* need not correspond to a computation of M. Assigning 1 to both $P_{0,0}$ & $P_{1,0}$ indicates that the tape head is at 2 distinct positions at time 0.

  The wff *f*(*u*) should impose restrictions on the variables to ensure that the interpretations of the variables are identical with those generated by the *truth assignment* obtained from a computation. Eight sets of wff are defined from *u* & the transitions of M. Seven of the eight families of wff are given directly in clause form.

15.8 The Satisfiability Problem

- Proof (Continued). The notation

$$\bigwedge_{i=1}^{k} v_i \qquad \bigvee_{i=1}^{k} v_i$$

represents the *conjunction* and *disjunction* of the literals $v_1, \ldots, v_k$, respectively.

A truth assignment that satisfies the set of *clauses* defined in (i) in the following table indicates that the TM is in a *unique state* at each time. Satisfying the first disjunction guarantees that at least one of the variables $Q_{i,k}$ holds. The pairwise negations specify that no two states are satisfied at the same time. This is most easily seen using the tautological equivalence of the disjunction $\neg A \vee B$ to the implication $A \Rightarrow B$ to transform the clauses $\neg Q_{i,k} \vee \neg Q_{i',k}$ into implications $Q_{i,k} \Rightarrow \neg Q_{i',k}$ which can be interpreted as asserting that if the TM is in state q_i at time k, then it is not also in $q_{i'}$, for any $i' \neq i$.

- Proof (Continued).

	Clause	Conditions	Interpretation (when satisfied)
i)	State		
	$\bigvee_{i=0}^{m} Q_{i,k}$	$0 \le k \le p(n)$	For each time k, M is in at least one state.
	$\neg Q_{i,k} \vee \neg Q_{i',k}$	$0 \le i < i' \le m$ $0 \le k \le p(n)$	M is in at most one state (not two different states at the same time).
ii)	Tape head position		
	$\bigvee_{j=0}^{p(n)} P_{j,k}$	$0 \le k \le p(n)$	For each time k, the tape head is in at least one position.
	$\neg P_{j,k} \vee \neg P_{j',k}$	$0 \le j < j' \le p(n)$ $0 \le k \le p(n)$	At most one position.
iii)	Symbols on tape		
	$\bigvee_{r=0}^{t} S_{j,r,k}$	$0 \le j \le p(n)$ $0 \le k \le p(n)$	For each time k and position j, position j contains at least one symbol.
	$\neg S_{j,r,k} \vee \neg S_{j,r',k}$	$0 \le j \le p(n)$ $0 \le r < r' \le t$ $0 \le k \le p(n)$	At most one symbol.
iv)	Initial conditions for input string $u = a_{r_1} a_{r_2} \ldots a_{r_n}$		
	$Q_{0,0}$ $P_{0,0}$ $S_{0,0,0}$		The computation begins reading the leftmost blank.
	$S_{1,r_1,0}$ $S_{2,r_2,0}$ ⋮ $S_{n,r_n,0}$		The string u is in the input position at time 0.
	$S_{n+1,0,0}$ ⋮ $S_{p(n),0,0}$		The remainder of the tape is blank at time 0.
v)	Accepting condition $Q_{m,p(n)}$		The halting state of the computations is q_m.

15.8 The Satisfiability Problem

- Proof (Continued). Since the computation of M with input of length n cannot access the tape beyond position $p(n)$, a TM configuration is completely defined by the *state*, *position* of the tape head, and the *contents* of the initial $p(n)$ positions of the tape.

  A truth assignment that satisfies the clauses in (i), (ii), and (iii) defines a TM configuration for each time between 0 and $p(n)$. The conjunction of the clauses (i) and (ii) indicates that the TM is in a unique state scanning a single tape position at each time. The clauses in (iii) ensure that the tape contains precisely one symbol in each position.

  A computation consists of a sequence of related configurations. Clauses whose satisfaction specifies the configuration at time 0 and links consecutive configurations are added. Initially, (i) the TM is in state q_0, (ii) the tape head scanning the leftmost position, (iii) the input on tape positions 1 to n, and the remaining tape squares blank. The satisfaction of the $p(n) + 2$ clauses in (iv) ensures the correct machine configuration at time 0.

15.8 The Satisfiability Problem

- Proof (Continued). Each subsequent configuration must be obtained from its successor by the application of a transition. Assume that the TM is in state q_i, scanning symbol *a* in position *j* at time *k*. The final three sets of wff are introduced to generate the permissible configurations at time *k* + 1 based on the transitions of M and the variables that define the configuration at time *k*.

  The effect of a transition on the tape is to rewrite the position scanned by the tape head. With the possible exception of position $P_{j,k}$, every tape position at time *k* + 1 contains the same symbol as at time *k*. Clauses must be added to the wff to ensure that the remainder of the tape is unaffected by a transition.

Clause	Conditions	Interpretation (when satisfied)
(vi) Tape consistency		
$\neg S_{j,r,k} \vee P_{j,k} \vee S_{j,r,k+1}$	$0 \le j \le p(n)$, $0 \le r \le t$, $0 \le k \le p(n)$	Symbols not at the position of the tape head are unchanged

15.8 The Satisfiability Problem

- Proof (Continued). (vi) is not satisfied if a change occurs to a tape position other than the one scanned by the tape head, since

$$\neg S_{j,r,k} \vee P_{j,k} \vee S_{j,r,k+1} \Leftrightarrow \neg P_{j,k} \Rightarrow (S_{j,r,k} \Rightarrow S_{j,r,k+1})$$

Now assume that for a given time k, the TM is in state q_i scanning symbol a, in position j. These features of a configuration are designated by the assignment of 1 to the Boolean variables $Q_{i,k}$, $P_{j,k}$, and $S_{j,r,k}$. The clause

a) $\neg Q_{i,k} \vee \neg P_{j,k} \vee \neg S_{j,r,k} \vee Q_{i',k+1}$ is satisfied only when $Q_{i',k+1}$ is true, which signifies that M has entered state $q_{i'}$, at time k+1. The symbol in position j at time k+1 and the tape head position are specified by the clauses

b) $\neg Q_{i,k} \vee \neg P_{j,k} \vee \neg S_{j,r,k} \vee S_{j,r',k+1}$, and

c) $\neg Q_{i,k} \vee \neg P_{j,k} \vee \neg S_{j,r,k} \vee P_{j+n(d),k+1}$, where $n(L) = -1$ and $n(R) = 1$

(a), (b) & (c) are satisfied by the transition $[q_{i'}, a_{r'}, d] \in \delta(q_i, a_r)$.

15.8 The Satisfiability Problem

- Proof (Continued). Except for q_m & q_{m-1}, the restrictions on M ensure that at least one transition is defined for each <state, symbol>.

The CNF formulas

$(\neg Q_{i,k} \vee \neg P_{j,k} \vee \neg S_{j,r,k} \vee Q_{i',k+1})$	New state
$(\neg Q_{i,k} \vee \neg P_{j,k} \vee \neg S_{j,r,k} \vee P_{j+n(d),k+1})$	New tape head position
$(\neg Q_{i,k} \vee \neg P_{j,k} \vee \neg S_{j,r,k} \vee S_{j,r',k+1})$	New symbol at position r

is constructed for every

$0 \leq k \leq p(n)$	time
$0 \leq i \leq m - 1$	non-halting state
$0 \leq j \leq p(n)$	tape head position
$0 \leq r \leq t$	tape symbol

where $[q_{i'}, a_{r'}, d] \in \delta(q_i, a_r)$, except when the position is 0 & the direction L is specified by the transition. For the exception when a transition causes the tape head to cross the leftmost cell of the tape, a special cause is encoded by the following wff:

15.8 The Satisfiability Problem

- Proof (Continued).

$(\neg Q_{i,k} \vee \neg P_{0,k} \vee \neg S_{0,r,k} \vee \mathrm{Q}_{m\text{-}1,k+1})$ Entering the reject state

$(\neg Q_{i,k} \vee \neg P_{0,k} \vee \neg S_{0,r,k} \vee \mathrm{P}_{0,k+1})$ Same tape head position

$(\neg Q_{i,k} \vee \neg P_{0,k} \vee \neg S_{0,r,k} \vee \mathrm{S}_{0,r,k+1})$ Same symbol at position r

for all transitions $[q_{i'}, a_{r'}, L] \in \delta(q_i, a_r)$.

Since M is nondeterministic, there may be several transitions that can be applied to a given configuration. The result of applying any of these alternatives is a permissible succeeding configuration in a computation.

Let *trans*(i, j, r, k) denote disjunction of the CNF formulas that represent the alternative transitions for a configuration at time k in state q_i, tape head position j, and tape symbol r. *Trans*(i, j, r, k) is satisfied only if the values of the variables at time k+1 represent a legitimate successor to the variables with time k.

15.8 The Satisfiability Problem

- Proof (Continued).

Formula	**Interpretation (when satisfied)**
vii) Generation of successor configuration $trans(i, j, r, k)$	Configuration k+1 follows from configuration k by the application of a transition

The formulas $trans(i, j, r, k)$ do not specify the actions to be taken when the TM is in state q_m or q_{m-1}. In this case, the subsequent configuration is identical to its predecessor.

Clause	**Interpretation (when satisfied)**
viii) Halted computation	
$(\neg Q_{i,k} \vee \neg P_{j,k} \vee \neg S_{j,r,k} \vee Q_{i,k+1})$	Same state
$(\neg Q_{i,k} \vee \neg P_{j,k} \vee \neg S_{j,r,k} \vee P_{j,k+1})$	Same tape head position
$(\neg Q_{i,k} \vee \neg P_{j,k} \vee \neg S_{j,r,k} \vee S_{j,r,k+1})$	Same symbol at position r

These clauses are built $\forall j, r, k$ in the legal range & $i = q_m, q_{m-1}$

15.8 The Satisfiability Problem

- Proof (Continued). Let $f'(u)$ be the conjunction of the wff constructed in (i) through (viii). When $f'(u)$ is satisfied by a *truth assignment* on V, the variables define the configurations of a computation of M that accepts the input string u. The clauses in (iv) specify that the configuration at time 0 is the initial configuration of a computation of M with input u. Each subsequent configuration is obtained from its successor by the result of the application of a transition. u is accepted by M since the satisfaction of (v) indicates that the final configuration contains the state q_m.

A CNF formula $f(u)$ can be obtained from $f'(u)$ by converting each formula *trans*(i, j, r, k) into CNF using the technique presented in Lemma 15.8.4 that follows. Lastly, we show that the transformation of a string $u \in \Sigma^*$ to $f(u)$ can be done in polynomial time.

The transformation of u to $f(u)$ consists of the construction of the clauses & the conversion of trans to CNF. The no. of clauses is a function of

15.8 The Satisfiability Problem

- Proof (Continued).

  i) the number of states m and the number of tape symbols t,

  ii) the length n of the input string u, and

  iii) the bound $p(n)$ on the length of the computation of M

  m and t obtained from M are independent of the input string. From the range of the subscripts, we see that the number of clauses is polynomial in $p(n)$. The development of $f(u)$ is completed with the transformation into CNF which, by Lemma 15.8.4, is polynomial in the number of clauses in the formulas *trans*(i, j, r, k).

  We have shown that the CNF formula can be constructed in a number of steps that grows *polynomially* with the length u. What is really needed is the representation of the formula that serves as input to a TM that solves the Satisfiability Problem. Any reasonable encoding, including the one developed in Theorem 15.8.2, requires only polynomial time to convert the high-level representation to the machine representation. □