CS423 Finite Automata & Theory of Computation

TTh 12:30 - 13:50 in Small Physics Lab 111 (section 1) TTh 9:30 - 10:50 in Blow 331 (section 2)

Prof. Weizhen Mao, wxmaox@wm.edu, wm@cs.wm.edu

General Information

- Office Hours: TTh 11:00 - 12:00 in 114 McGl and W 2:30 - 3:00 on zoom or by email
- Grader: TBD for section 1 (office hour TBD on BB)
- Grader: TBD for section 2 (office hour TBD on BB)
- Textbook: Intro to the theory of computation (any edition), Michael Sipser. An e-book in PDF maybe available online.
- Prerequisites/background: Linear algebra, Data structures and algorithms, and Discrete math

Complexity Theory:

- Computability Theory is the study of what can or cannot be computed by a TM/algorithm, among all problems.
- Complexity Theory is the study of what can or cannot be computed **efficiently** by a TM/algorithm, among all decidable/solvable problems.
- For the set of all solvable problems, it is further classified into various **complexity classes** based on the efficiency of algorithms solving these problems.
- Complexity Theory is the study of the definition and properties of these classes.

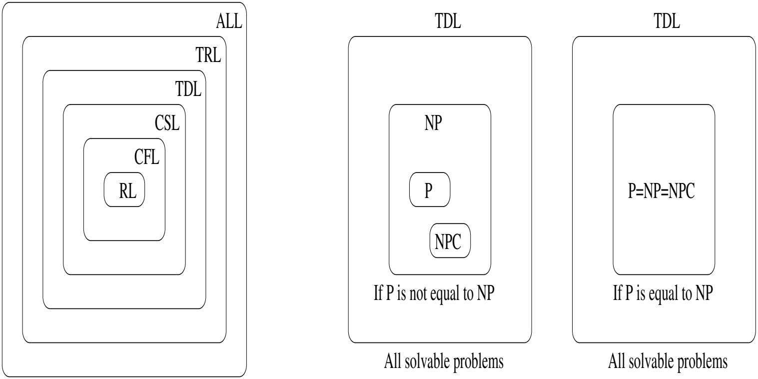

Figure 1: Three complexity classes if P=NP or P$\neq$NP

11.1 The class of P (*Sipser 7.2*)

- Definition: **P** is the class of problems solvable in polynomial time (number of steps) by deterministic TMs. Polynomial $O(n^c)$, where n is input size and c is a constant. Problems in **P** are ”tractable” (not so hard).
- Why use polynomial as the criterion?
  - If a problem is not in **P**, it often requires unreasonably long time to solve for large-size inputs.
  - **P** is independent of all models of computation, except nondeterministic TM.
- Problems in **P**: Sorting, Searching, Selecting, Minimum Spanning Tree, Shortest Path, Matrix Multiplication, etc.
- Review of asymptotic notation: O, Ω, Θ
- Examples of polynomial and polylog functions: $O(1)$, $O(n)$, $O(n^2)$, $O(n^d)$, $O(\log n)$, $O((\log n)^c)$, $O(n^3 \log n)$, $O(n^c(\log n)^d)$

11.2 The class of NP (*Sipser 7.3*)

- An NTM is an unrealistic (unreasonable) model of computing which can be simulated by other models with an exponential loss of efficiency.
  It is a useful concept that has had great impact on the theory of computation.
- NTM $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where $\delta : Q \times \Gamma \to 2^P$ for $P = Q \times \Gamma \times \{L, R\}$.
- $\delta(q, X)$ is a set a moves. Which one to choose? This is nondeterminism. The computation can be illustrated by a tree, with each node representing a configuration.

- Nondeterminism can be viewed as a kind of parallel computation wherein multiple independent processes or threads can be running concurrently.
  When a nondeterministic machine splits into several choices, that corresponds to a process forking into several children, each then proceeding separately. If at least one process accepts, then the entire computation accepts.
- Time complexity of nondeterministic TMs (NTMs): Let N be an NTM that is a decider (where all computation paths halt in the tree for any input). The time complexity of N, $f(n)$, is the maximum number of steps that N uses on any computation path for any input of length n. In other words, $f(n)$ is the maximum height of all computation trees for all input of length n.

- An unreasonable model of computation:

  Theorem; Every $T(n)$-time multi-tape TM has an equivalent $O(T^2(n))$-time single-tape TM.

  Theorem: Every $T(n)$-time single-tape NTM has an equivalent $O(2^{O(T(n))})$-time single-tape DTM.
- Definition: **NP** is the class of problems solvable in polynomial time by nondeterministic TMs.
- Another definition of nondeterministic TMs (algorithms):
  - Guessing phase: Guess a solution (always on target).
  - Verifying phase: Verify the solution.

Example: TSP (DEC) is in **NP**.

INSTANCE: An edge-weighted graph $G(V,E,w)$ and a bound $B \geq 0$.

QUESTION: Is there a tour (a cycle that passes through each node exactly once) in G with total weight no more than B?
Define the following NTM N to solve TSP in polynomial time.

NTM $N =$"On input $< G, B >$

1. Nondeterministically **guess** a tour T $\quad O(|V|)$
2. **Verify** if T includes every node once $\quad O(|V|)$
3. Compute $sum \leftarrow \sum_{e \in T} w(e)$ $\quad O(|V|)$
4. **Verify** if $sum \leq B$. If true, answer **yes**; else **No** $\quad O(1)$

Note 1: The time complexity of N is $O(|V|)$.
Note 2: If the answer to the QUESTION is "Yes", N guarantees that the right T will be guessed in step 1.

Note 3: The acceptance of an input by a nondeterministic machine is determined by whether there is an accepting computation among all, possibly exponentially many, computations.

In the above proof, if there is a solution, i.e., a tour with total weight no more than B, it will always be generated by the Turing machine. This is like that a nondeterministic machine has a guessing power.

A tour can only be found by a deterministic machine in exponential time, however, it can be found by a nondeterministic machine in just linear steps. Any nondeterministic proof should always contain two stages: Guessing and verifying.

What needs to be guessed? What needs to be verified? What is the time complexity?

Example: Graph Coloring (GC) is in **NP**.

INSTANCE: Graph $G = (V, E)$, and $B \geq 0$

QUESTION: Is there a coloring scheme of the nodes that uses no more than B colors such that no two nodes connected by an edge are given the same color?

NTM $N =$"On input $< G, B >$

1. Guess a coloring scheme (in polynomial time) $c : V \rightarrow C$
2. Verify if (1) $|C| \leq B$ and (2) $\forall (u, v) \in E, c(u) \neq c(v)$
3. if true, answer **yes**; else answer **no**

So we have a **nondeterministic** algorithm (or TM) that **guesses** a coloring scheme (or function) and **verifies** that (1) for any $(u, v) \in E$, $c(u) \neq c(v)$ and that (2) the number of colors used is no more than B, and further, all these can be done in **polynomial time** of $O(|V|) + O(|E|)$. So GC is in **NP**.

- Theorem: **P**$\subseteq$**NP**.(Two possibilities: **P** $\subset$ **NP** or **P** $=$ **NP**)
  Any deterministic TM is a special case of nondeterministic TMs.
- Theorem: Any Π $\in$**NP** can be solved by a deterministic TM in time $O(c^{p(n)})$ for some $c > 0$ and polynomial $p(n)$.
- Open problem: **P**$=$**NP**?
  The west wall bricks on the CS building at Princeton,1989:
  x1010000x
  x0111101x
  x1001110x
  x1010000x
  x0111111x

- Definition of **Polynomial Reduction** $\leq_p$ (cf. $\leq_m$ and $\leq$)
  Let Π_1 and Π_2 be two decision problems, and $\{I_1\}$ and $\{I_2\}$ be sets of instances for Π_1 and Π_2, respectively.
  We say there is a polynomial reduction from Π_1 to Π_2, or $\Pi_1 \leq_p \Pi_2$, if there is $f : \{I_1\} \to \{I_2\}$ such that
  (1) f can be computed in polynomial time and
  (2) I_1 has a "yes" solution if and only if $f(I_1)$ has a "yes" solution.
- Theorem: If $\Pi_1 \leq_p \Pi_2$, then $\Pi_2 \in$**P** implies $\Pi_1 \in$**P**.
- Theorem: If $\Pi_1 \leq_p \Pi_2$ and $\Pi_2 \leq_p \Pi_3$, then $\Pi_1 \leq_p \Pi_3$.
- *Remark:* $\leq_p$ means "no harder than".

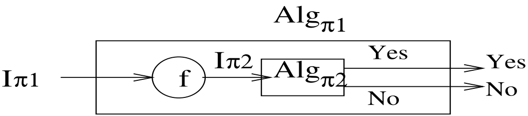

Figure 2: Polynomial reduction $\Pi_1 \leq_p \Pi_2$

11.4 The class of NPC (*Sipser 7.4*)

- Definition 1: **NPC** (**NP**-complete) is the class of the hardest problems in **NP**
- Definition 2: $\Pi \in$**NPC** if $\Pi \in$**NP** and $\forall \Pi' \in$**NP**, $\Pi' \leq_p \Pi$.
- Definition 3: $\Pi \in$**NPC** if $\Pi \in$**NP** and $\exists \Pi' \in$**NPC** such that $\Pi' \leq_p \Pi$
- Theorem: If $\exists \Pi \in$**NPC** such that $\Pi \in$**P**, then **P**$=$**NP**.
- Theorem: If $\exists \Pi \in$**NPC** such that $\Pi \notin$ **P**, then **P**$\neq$**NP**.

Some most important classes: Definitions and proofs

- **P**: class of problems solvable in polynomial-time by DTM. To prove $\Pi \in$**P**, design a polynomial-time algorithm.
- **NP**: class of problems solvable in polynomial-time by NTM. To prove $\Pi \in$**NP**, design a polynomial-time nondeterministic algorithm of two steps: guess and verify.
- **NPC**: class of all hardest problems in **NP**. To prove $\Pi \in$**NPC**, prove (1) $\Pi \in$**NP** and (2) $\exists \Pi' \in$ **NPC** s.t. $\Pi' \leq_p \Pi$
- **NP-hard**: A problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time. (Note X does not need to be in **NP**)

Possible relations among P, NP, NP-complete, NP-hard:
P$\subset$**NP**, **NP**$\cap$ **NP-hard** = **NP-complete**, **P**$\cup$ **NP-complete**$=\emptyset$

Satisfiability (SAT):

INSTANCE: A boolean formula ϕ in CNF with variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$

QUESTION: Is ϕ satisfiable? (Is there a truth assignment A to $x_1, \ldots, x_n$ such that ϕ is true?)

$L_{SAT} = \{< \phi > | \exists A \text{ that satisfies } \phi\}$

Example of an instance for SAT:

- Variables: x_1, x_2, x_3, x_4
- Literals: Any variables and their negations, such as x_1, $\overline{x_3}$
- Clauses: $c_1 = x_1 \vee \overline{x_2} \vee x_3$, $c_2 = x_1 \vee x_2$, $c_3 = \overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4$
- Function/formula: $\phi = c_1 \wedge c_2 \wedge c_3$
- The instance ϕ is T by assignment $x_1 = T, x_2 = x_3 = x_4 = F$.
  Note: Many assignments satisfy ϕ, but we only need one.

Cook's Theorem: SAT$\in$**NPC.** (Need to prove (1) SAT$\in$**NP** and (2) $\forall \Pi \in$**NP**, $\Pi \leq_p$SAT.)

First Step: How to prove SAT is in **NP**?
NTM $N =$ "On input $< \phi >$ in CNF

1. Guess a truth assignmnet A $\qquad O(n)$
2. Verify if $\phi = T$ under A $\qquad O(n+m)$
3. If T, **accept**; else **reject**"

SAT is solvable by a NTM in polynomial time, thus in **NP.**

Second step: How to prove $\forall \Pi \in$**NP**, $\Pi \leq_p$SAT, or equivalently, for any polynomial-time NTM M, $L(M) \leq_p L_{SAT}$?

Will not discuss this proof. But if interested, go to the final few pages of this slide set for details.

11.5 NP-complete problems(Sipser 7.5, pp.310-322)

How to prove Π_2 is **NP**-complete:

- Show that $\Pi_2 \in$**NP**.
- Choose a known **NP**-complete Π_1.
- Construct a reduction f from Π_1 to Π_2.
- Prove that f is a polynomial reduction by showing (1) f can be computed in polynomial time and (2) $\forall I_1$ for Π_1, I_1 is a yes-instance for Π_1 if and only if $f(I_1)$ is a yes-instance for Π_2.

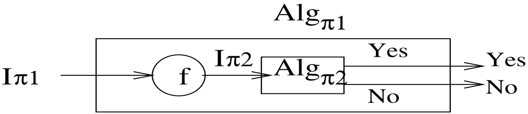

Figure 3: Polynomial reduction $\Pi_1 \leq_p \Pi_2$

Seven basic **NP**-complete problems.

- 3-Satisfiability (3SAT): (Reduced from SAT)
  INSTANCE: A formula α in CNF with each clause having three literals.
  QUESTION: Is α satisfiable?
- 3-Dimensional Matching (3DM): (Reduced from 3SAT)
  INSTANCE: $M \subseteq X \times Y \times Z$, where X, Y, Z are disjoint and of the same size.
  QUESTION: Does M contain a matching, which is $M' \subseteq M$ with $|M'| = |X|$ such that no two triples in M' agree in any coordinate?
- PARTITION: (Reduced from 3DM)
  INSTANCE: A finite set A of numbers.
  QUESTION: Is there $A' \subseteq A$ such that $\sum_{a \in A'} a = \sum_{a \in A - A'} a$?

- Vertex Cover (VC): (Reduced from 3SAT)
  INSTANCE: A graph $G = (V, E)$ and $0 \leq k \leq |V|$.
  QUESTION: Is there a vertex cover of size $\leq k$, where a vertex cover is $V' \subseteq V$ such that $\forall (u, v) \in E$, either $u \in V'$ or $v \in V'$?
- Hamiltonian Circuit (HC): (Reduced from VC)
  INSTANCE: A graph $G = (V, E)$.
  QUESTION: Does G have a Hamiltonian circuit, i.e., a tour that passes through each vertex exactly once?
- CLIQUE: (Reduced from VC)
  INSTANCE: A graph $G = (V, E)$ and $0 \leq k \leq |V|$.
  QUESTION: Does G contain a clique (complete subgraph) of size $\geq k$?

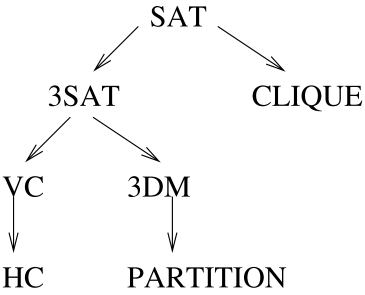

Figure 4: Seven basic NP-complete problems

Example to prove reduction: KNAPSACK Problem

INSTANCE: $U = \{u_1, \ldots, u_n\}$, W (max weight knapsack holds), functions $w : U \to R^+$ and $v : U \to R^+$, bound $B \geq 0$.
QUESTION: Is there $U' \subseteq U$ s.t. $\sum_{u_i \in U'} w(u_i) \leq W$, and $\sum_{u_i \in U'} v(u_i) \geq B$?

Show PARTITION$\leq_p$ KNAPSACK.
For any instance $\{a_1, \ldots, a_n\}$ in PARTITION, define the following instance for KNAPSACK:

- $U = \{u_1, \ldots, u_n\}$
- $w(u_i) = a_i$ and $v(u_i) = a_i$, for all i
- $W = B = \frac{1}{2} \sum_{i=1}^{n} a_i$

Show that (1) the above construction can be done in polynomial time and (2) there is a partition $A' \subseteq A$ iff there is a subset U' s.t. $\sum_{u_i \in U'} w(u_i) \leq \frac{1}{2} \sum_{i=1}^{n} a_i$ and $\sum_{u_i \in U'} v(u_i) \geq \frac{1}{2} \sum_{i=1}^{n} a_i$.

Example to prove reduction: Hitting Set (HS) problem

INSTANCE: Set S, collection C of subsets of S, $K \geq 0$
QUESTION: Does S contain a HS S' of size $\leq K$? (S' is a HS if S' has at least one element from each subset in C.) $VC \leq_p HS$.
Instance for VC: $G = (V, E)$ and $B \geq 0$
Instance for HS: $S = V$, $C = \{\{u, v\} | \forall e = (u, v) \in E\}$, $K = B$
Goal: G has a VC of size $\leq B$ iff S has a HS of size $\leq K$.

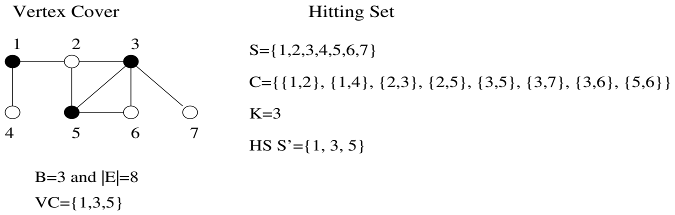

Figure 5: An example of reduction from VC to HS

Review of Automata Theory

- RL, CFL, DFA, NFA, PDA, Regular expressions
- Closure properties for the above sets of languages
- Pumping Lemmas for RL
- Chomsky Language Hierarchy
- Prove by contradiction

Review of Computability Theory

- TDL: How to prove a language is TDL
- Closure properties of TDLs: union, intersection, concatenation, star, complement
- A TM that decides (accept, reject)
- TRL: How to prove a language is TRL
- The closure properties of TRLs: union, intersection, concatenation, star, homomorphism
- TM to decide/accepts/recognizes. NTM to guess/verify
- TD, non-TD, TR, non-TR (How to prove)
- Closure properties for the above sets of languages
- A language and its complement: Three scenarios (both TD, one TR but non-TD other non-TR, both non-TR)
- Important languages: A_D, A_{TM}, $HALT_{TM}$, etc.

- Reduction $A \leq B$ is to show any TM that decides B can be used to define a TM that decides A. (A is no harder than B or B is at least as hard as A.)
- P, NP, NPC, NP-Hard (relation based on P$\neq$NP, P=NP)
- Prove a DEC is in NP.
- Three classes: P, NP, NPC (Also NP-hard)
- Polynomial reduction $A \leq_p B$ is to show any algorithm that solves B can be used to define an algorithm that solves A. (A is no harder than B or B is at least as hard as A.)
- Important NP-complete problems: SAT, 3SAT, VC, HC, PARTITION, 3DM, CLIQUE, COLOR, KNAPSACK, HS
- **Test of your understanding of the complexity classes**
  - If $\Pi_1 \leq_p \Pi_2$ and $\Pi_1 \in$**NP**, is $\Pi_2 \in$**NP**?
  - If $\Pi_1 \leq_p \Pi_2$ and $\Pi_1, \Pi_2 \in$**NPC**, is $\Pi_2 \leq_p \Pi_1$?
  - If $\Pi_1 \leq_p \Pi_2$ and $\Pi_1 \notin$**NP**, is $\Pi_1 \in$**P**?
  - If $\Pi_1 \leq_p \Pi_2$ and $\Pi_2 \leq_p \Pi_1$, then $\Pi_1, \Pi_2 \in$ **NPC**.
  - If $\Pi_1, \Pi_2 \in$ **NPC**, then $\Pi_1 \leq_p \Pi_2$, and $\Pi_2 \leq_p \Pi_1$.

Some sample problems

- The universal language, A_{TM}, is a proper (non-equal) subset of the halting language, $HALT_{TM}$.
- The Post Correspondence Problem is decidable for the unary alphabet.
- If L is TR but non-TD, then $\overline{L}$ is non-TR.

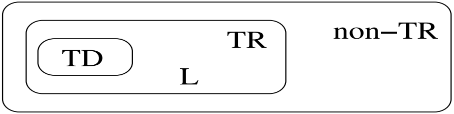

Figure 6: Venn diagram for TD, TR, and non-TR

- If A is non-TD, $A \leq C$, $D \leq C$, then D must be non-TR.
  F (because D may be TD or TR)

end

A proof that 3SAT is NP-complete:
First, 3SAT is obvious in **NP**.
Next, we show that SAT$\leq_p$3SAT.
Given any instance of SAT, $f(x_1,\ldots,x_n) = c_1 \wedge \cdots \wedge c_m$, where c_i is a disjunction of literals. To construct an instance for 3SAT, we need to convert any c_i to an equivalent c_i', a conjunction of clauses with exactly 3 literals.
Case 1. If $c_i = z_1$ (one literal), define y_i^1 and y_i^2. Let $c_i' =$ $(z_1 \vee y_i^1 \vee y_i^2) \wedge (z_1 \vee y_i^1 \vee \neg y_i^2) \wedge (z_1 \vee \neg y_i^1 \vee y_i^2) \wedge (z_1 \vee \neg y_i^1 \vee \neg y_i^2)$.
Case 2. If $c_i = z_1 \vee z_2$ (two literals), define y_i^1. Let $c_i' = (z_1 \vee z_2 \vee y_i^1) \wedge (z_1 \vee z_2 \vee \neg y_i^1)$.
Case 3. If $c_i = z_1 \vee z_2 \vee z_3$ (three literals), let $c_i' = c_i$.

Case 4. If $c_i = z_1 \vee z_2 \vee \cdots \vee z_k$ ($k > 3$), define $y_i^1, y_i^2, \ldots, y_i^{k-3}$. Let $c_i' = (z_1 \vee z_2 \vee y_i^1) \wedge (\neg y_i^1 \vee z_3 \vee y_i^2) \wedge (\neg y_i^2 \vee z_4 \vee y_i^3) \wedge \cdots \wedge (\neg y_i^{k-3} \vee z_{k-1} \vee z_k)$.
If c_i is satisfiable, then there is a literal $z_l = T$ in c_i. If $l = 1, 2$, let $y_i^1, \ldots, y_i^{k-3} = F$. If $l = k-1, k$, let $y_i^1, \ldots, y_i^{k-3} = T$. If $3 \leq l \leq k-2$, let $y_i^1, \ldots, y_i^{l-2} = T$ and $y_i^{l-1}, \ldots, y_i^{k-3} = F$. So c_i' is satisfiable.
If c_i' is satisfiable, assume $z_l = F$ for all $l = 1, \ldots, k$. Then $y_i^1, \ldots, y_i^{k-3} = T$. So the last clause $(\neg y_i^{k-3} \vee z_{k-1} \vee z_k) = F$. Therefore, c_i' is not satisfiable. Contradiction.
The instance of 3SAT is therefore $f'(x_1, \ldots, x_n, \ldots) = c_1' \wedge \cdots \wedge c_m'$, and f is satisfiable if and only if f' is satisfiable.

Cook's Theorem: SAT is **NP**-complete.

Proof. SAT is clearly in **NP** since a NTM exists that guesses a truth assignment and verifies its correctness in polynomial time. Now we wish to prove $\forall \Pi \in$**NP**, $\Pi \leq_p$SAT, or equivalently, for any polynomial-time NTM M, $L(M) \leq_p L_{SAT}$.

For any NTM M, assume $Q = \{q_0, q_1(\text{accept}), q_2(\text{reject}), \ldots, q_r\}$ and $\Gamma = \{s_0, s_1, s_2, \ldots, s_v\}$. Also assume that the time is bounded by $p(n)$, where n is the length of the input.

We wish to prove that there is a function $f_M : \Sigma^* \to \{\text{instances of SAT}\}$ such that $\forall x \in \Sigma^*$, $x \in L(M)$ iff $f_M(x)$ is satisfiable. In other words, we wish to use a Boolean expression $f_M(x)$ to describe the computation of M on x.

Variables in $f_M(x)$:
— State: $Q[i,k]$. M is in q_k after the ith step of computation (at time i).
— Head: $H[i,j]$. Head points to tape square j at time i.
— Symbol: $S[i,j,l]$. Tape square j contains s_l at time i.
(Assume the tape is one-way infinite and the leftmost square is labeled with 0.)
For example, initially $i = 0$. Assume the configuration is $q_0 abba$. Let $s_0 = B$, $s_1 = a$, and $s_2 = b$. Therefore, we set the following Boolean variables to be true: $Q[0,0]$, $H[0,0]$, $S[0,0,1]$, $S[0,1,2]$, $S[0,2,2]$, $S[0,3,1]$ and $S[0,j,0]$ for $j = 4,5,\ldots$. A configuration defines a truth assignment, but not vice versa.

Clauses in $f_M(x)$:

— At any time i, M is in exactly one state.

$Q[i,0] \vee \cdots \vee Q[i,r]$ for $0 \leq i \leq p(n)$.

$\neg Q[i,k] \vee \neg Q[i,k']$ for $0 \leq i \leq p(n)$ and $0 \leq k < k' \leq r$.

— At any time i, head is scanning exactly one square.

$H[i,0] \vee \cdots \vee H[i,p(n)]$ for $0 \leq i \leq p(n)$.

$\neg H[i,j] \vee \neg H[i,j']$ for $0 \leq i \leq p(n)$ and $0 \leq j < j' \leq p(n)$.

— At any time i, each square contains exactly one symbol.

$S[i,j,0] \vee \cdots \vee S[i,j,v]$ for $0 \leq i \leq p(n)$ and $0 \leq j \leq p(n)$.

$\neg S[i,j,l] \vee \neg S[i,j,l']$ for $0 \leq i \leq p(n)$, $0 \leq j \leq p(n)$ and $0 \leq l < l' \leq v$.

— At time 0, M is in its initial configuration. Assume $x = s_{l_1} \cdots s_{l_n}$.

$Q[0,0]$.

$H[0,0]$.

$S[0,0,l_1], \ldots, S[0,n-1,l_n]$.

$S[0,j,0]$ for $n \leq j \leq p(n)$.

— By time $p(n)$, M has entered q_1 (accept)). (If M halts in less than $p(n)$ steps, additional moves can be included in the transition function.)

$Q[p(n),1]$.

— Configuration at time $i \rightarrow$ configuration at time $i+1$. Assume $\delta(q_k, s_l) = (q_{k'}, s_{l'}, D)$, where $D = -1, 1$.

If the head does not point to square j, symbol on j is not changed from time i to time $i+1$.

$H[i,j] \vee \neg S[i,j,l] \vee S[i+1,j,l]$ for $0 \leq i \leq p(n)$, $0 \leq j \leq p(n)$, and $0 \leq l \leq v$.

If the current state is q_k, the head points to square j which contains symbol s_l, then changes are made accordingly.

$\neg H[i,j] \vee \neg Q[i,k] \vee \neg S[i,j,l] \vee H[i+1, j+D]$,

$\neg H[i,j] \vee \neg Q[i,k] \vee \neg S[i,j,l] \vee Q[i+1, k']$, and

$\neg H[i,j] \vee \neg Q[i,k] \vee \neg S[i,j,l] \vee S[i+1, j, l']$, for $0 \leq i \leq p(n)$, $0 \leq j \leq p(n)$, $0 \leq k \leq r$, and $0 \leq l \leq v$.

Let $f_M(x)$ be the conjunction of all the clauses defined above. Then $x \in L(M)$ iff there is an accepting computation of M on x iff $f_M(x)$ is satisfiable. f_M can be computed in polynomial time since $|f_M(x)| \leq$ (number of clauses) $*$(number of variables)$= O(p(n)^2) * O(p(n)^2) = O(p(n)^4)$.
So there is a polynomial reduction from any language in **NP** to SAT. So SAT is **NP**-complete.