Handout

wu

March 4, 2022

Contents

1 Chap 1: The computational model

1.0.1 Efficiency and Running Time

Definition 1.1. A TM M is described by a tuple (Γ, Q, δ) containing

- A finite set Γ of the symbols that M's tapes can contain. We assume that Γ contains a designated "blank" symbol, denoted $\square$; a designated "start" symbol, denoted $\rhd$; and the numbers 0 and 1. We call Γ the **alphabet** of M

- A finite set Q of possible states M' register can be in. We assume that Q contains a designated start state, denoted q_{start}, and a designated halting state, denoted q_{halt}

- A function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\text{L,S,R}\}^k$, where $k \geq 2$, describing the rules M use in performing each step. This function is called the **transition function** of M

Definition 1.2 (Computing a function and running time)**.** Let $f : \{0,1\}^* \to \{0,1\}$ and let $T : \mathbb{N} \to \mathbb{N}$ be some functions, and let M be a Turing machine. We say that M **computes** f if for every $x \in \{0,1\}^*$, whenever M is initialized to the start configuration on input x, then it halts with $f(x)$ written on its output tape. We say M **computes** f **in** $T(n)$**-time** if its computation on every input x requires at most $T(|x|)$ steps

A function $T : \mathbb{N} \to \mathbb{N}$ is **time constructible** if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto \llcorner T(|x|) \lrcorner$ in time $T(n)$. ($\llcorner T(|x|) \lrcorner$ denotes the binary representation of the number $T(|x|)$). The restriction $T(n) \geq n$ is to allow the algorithm time to read its input.

Proposition 1.3. *For every $f : \{0,1\}^* \to \{0,1\}$ and a time-constructible bT : $\mathbb{N} \to \mathbb{N}$, if f is computable in time $T(n)$ by a TM M using alphabet Γ, then it's computable in time $4\log|\Gamma|T(n)$ by a TM M using the alphabet $\{0,1,\Box,\rhd\}$.*

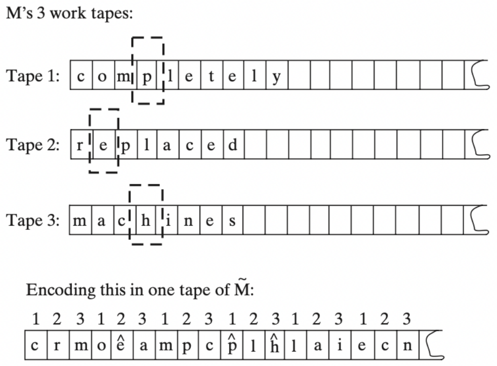
Proof. Let M be a TM with alphabet Γ, k tapes and state set Q that computes the function f in $T(n)$ times. We describe an equivalent TM $\tilde{M}$ computing f with alphabet $\{0,1,\Box,\rhd\}$, k tapes and a set Q' of states.

One can encode any member of Γ using $\log|\Gamma|$ bits. Thus each of $\tilde{M}$'s work tapes will simply encode one of M's tapes: For every cell in M's tape we will have $\log|\Gamma|$ cells in the corresponding tape of $\tilde{M}$

To simulate one step of M, the machine $\tilde{M}$ will 1. use $\log|\Gamma|$ steps to read from each tape the $\log|\Gamma|$ bits encoding of a symbol of Γ 2. use its state register to store the symbols read 3. use M's transition function to compute the symbols M writes and M's new state given this information 4. store this information in its state register 5. use $\log|\Gamma|$ steps to write the encodings of these symbols on its tapes □

Proposition 1.4. *Define a single-tape Turing machine to be a TM that has only one read-write tape. For every $f : \{0,1\}^* \to \{0,1\}$ and time-constructible $T : \mathbb{N} \to \mathbb{N}$ if f is computable in time $T(n)$ by a TM M using k tapes, then it is computable in time $5kT(n)^2$ by a single-tape TM M*

Proof. The TM $\tilde{M}$ encodes k tapes of M on a single tape by using locations $1, k+1, 2k+1, ...$ to encode the first tape, locations $2, k+2, 2k+2, ...$ to encode the second tape etc. For every symbol a in M's alphabet, $\tilde{M}$ will contain both the symbol a and the symbol $\hat{a}$. In the encoding of each tape, exactly one symbol will be of the "^ type", indicating that the corresponding head of M is positioned in that location. $\tilde{M}$ will not touch the first $n+1$ locations of its tape (where the input is located) but rather start by taking $O(n^2)$ steps



to copy the input bit by bit into the rest of the tape, while encoding it in the above way. □

Remark (Oblivious Turing machines). One can ensure that the proof of Proposition 1.4 yields a TM $\tilde{M}$ with the following property: its head movements do not depend on the input but only depend on the input length. That is, every input $x \in \{0,1\}^*$ and $i \in \mathbb{N}$, the location of each of M's at the ith step of execution on input x is only a function of $|x|$ and i. A machine with this property is called **oblivious**.

Exercise 1.0.1. Define a TM M to be **oblivious** if its head movements do not depend on the input but only on the input length. That is, M is oblivious if for every input $x \in \{0,1\}^*$ and $i \in \mathbb{N}$, the location of each of M's heads at the ith step of execution on input x is only a function of $|x|$ and i. Show that for every time-constructible $T : \mathbb{N} \to \mathbb{N}$, if $L \in \mathbf{DTIME}(T(n))$, then there is an oblivious TM that decides L in time $O(T(n)^2)$. Furthermore, show that there is such a TM that uses only **two tapes**: one input tape and one work/output tape

Proof. We can construct an oblivious TM M′ such that L(M)=L(M′) using a technique similar to that in the proof of Claim 1.6.

Observe that (1) after n steps, M's tape heads will all be in the position range [-n, n], and (2) since M runs in time O(T(n)), there exists some constant k such that for every input n, M runs n fewer than k*T(n) steps.

First, add two additional work tapes T_1 and T_2. The first thing M′ does is count the length of the input and write it to T_1. Next, write k*T(n) to T_2. Since T is time-constructible, and we have the value of 'n' on T_1, this takes time at most T(n).

Next, we simulate the execute of M in an oblivious way. We simulate each step i of M's execution as follows:

- Start every head t position -i. Move it right to position +i, finding the tape-head-location marker along the way (See Claim 1.6). We note the tapes's movement, writes, and state transition and then return to -(i+1) to begin the next step. If a tape's head is to move right, do so on the right pass, otherwise remember and when we see the marker on the way back, move it at that time. The write and state change can be made on the first pass.
- Simultaneously, decrement the value on T_2.

When T_2 is 0, halt.

The proof of language equality is essentially the same as in Claim 1.6, and obliviousness is apparent. Waiting k*T(n) simulated steps to halt guarantees that all machines of this input length would have halted by then.

Time anlaysis:

1. Writing T_1, incrementing one-by-one, takes time n*log(n). (For each symbol in the input, we may have to manipulate up to log(n) inputs of the length-so-far's binary representation to increment the value.)
2. Writing T_2 takes time T(n), by the definition of time-constructible
3. Each simulation step, the tapes move up to O(T(n)) steps. Keeping track of -i and i can be done on another tape that simply counts in unary, with the head moving across it to count out the correct distance.
4. Decrementing T_2 takes O(log(k*T(n))) = O(log(T(n))) steps
5. Multiplying c+d by the O(T(n)) "simulated steps" gets us O(T(n))*O(T(n) + log(T(n))) = O(T(n)*T(n) + log(T(n))*T(n)), and since and log(T(n)) < T(n), this simulation runs in O(T(n)**2)

It remains to show that this simulation can be done in as few as 2 tapes. To do this, we interleave all tapes other than the input tape in the standard way.

- Writing T_1 is only a constant factor slower.
- Similarly with writing T_2, since these are performed sequentially with no other non-input tapes active.

- Each simulation step is also only a constant-factor slower, since we can run over all the interleaved tapes at once, and the T_2 decrement is performed sequentially afterward.

□

Proposition 1.5. *Define a bidirectional TM to be a TM whose tapes are infinite in both directions. For every $f : \{0,1\}^* \to \{0,1\}^*$ and time-constructible $T : \mathbb{N} \to \mathbb{N}$ if f is computable in time $T(n)$ by a directional TM M, then it is computable in time $4T(n)$ by a standard (undirectional) TM $\tilde{M}$*

1.0.2 The Class P

A **complexity class** is a set of function that can be computed within given resource bounds.

We say that a machine **decides** a language $L \subseteq \{0,1\}^*$ if it computes the function $f_L : \{0,1\}^* \to \{0,1\}$ where $f_L(x) = 1 \Leftrightarrow x \in L$

Definition 1.6. Let $T : \mathbb{N} \to \mathbb{N}$ be some function. A language L is in **DTIME**$(T(n))$ iff there is a Turing machine that runs in $c\dot{T}(n)$ for some constant $c > 0$ and decides L.

The D in **DTIME** refers to "deterministic".

Definition 1.7. $\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c)$

2 Chap 2: NP and NP completeness

2.0.1 The Class NP

Definition 2.1. A language $L \subseteq \{0,1\}^*$ is in **NP** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M (called the **verifier** for L) s.t. for every $x \in \{0,1\}^*$

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$ then we call u a **certificate** for x

Example 2.1 (`INDSET` $\in$ **NP**)**.** By representing the possible invitees to a dinner party with the vertices of a graph having an edge between any two people who don't get along. The dinner party computational problem becomes the problem of finding a maximum sized **independent set** (set of vertices without any common edges) in a given graph. The corresponding language is

$$\texttt{INDSET} = \{\langle G, k\rangle : \exists S \subseteq V(G) \text{ s.t. } |S| \geq k \text{ and } \forall u, v \in S, \overline{uv} \notin E(G)\}$$

Consider the following polynomial-time algorithm M: Given a pair $\langle G, k\rangle$ and a string $u \in \{0,1\}^*$, output 1 iff u encodes a list of k vertices of G s.t. there is no edge between any two members of the list. Note that if n is the number of vertices in G, then a list of k vertices can be encoded using $O(k \log n)$ bits, where n is the number of vertices in G. Thus u is a string of at most $O(n \log n)$ bits, which is polynomial in the size of the representation of G.

Proposition 2.2. *Let **EXP** $= \bigcup_{c>1}$ **DTIME**(2^{n^c}). Then **P** $\subseteq$ **NP** $\subseteq$ **EXP***

Proof. **P** $\subseteq$ **NP**. Suppose $L \in$ **P** is decided in polynomial-time by a TM N. Then we take N as the machine M and make $p(x)$ the zero polynomial

NP $\subseteq$ **EXP**. We can decide L in time $2^{O(p(n))}$ by enumerating all possible n and using M to check whether u is a valid certificate for the input x. Note that $p(n) = O(n^c)$ for some $c > 1$, the number of choices for u is $2^{O(n^c)}$. □

NP stands for **nondeterministic polynomial time**.

NDTM has **two** transition function δ_0 and δ_1, and a special state denoted by q_{accept}. When an NDTM M computes a function, we envision that at each computational step M makes an arbitrary choice at to which of its two transition functions to apply. For every input x, we say that $M(x) = 1$ if there **exists** some sequence of this choices that would make M reach q_{accept} on input x. We say that M runs in $T(n)$ time if for every input $x \in \{0,1\}^*$ and every sequence of nondeterministic choices, M reaches the halting state or q_{accept} within $T(|x|)$ steps

Definition 2.3. For every function $f : \mathbb{N} \to \mathbb{N}$ and $L \subsetneq \{0,1\}^*$ we say that $L \in$ **NTIME**$(T(n))$ if there is a constant $c > 0$ and a $c\dot{T}(n)$-time NDTM M s.t. for every $x \in \{0,1\}^*$, $x \in L \Leftrightarrow M(x) = 1$

Theorem 2.4. ***NP*** $= \bigcup_{c \in \mathbb{N}}$ ***NTIME***(n^c)

Proof. The main idea is that the sequence of nondeterministic choices made by an accepting computation of an NDTM can be viewedas a certificate that the input is in the language, and vice versa

Suppose $p : \mathbb{N} \to \mathbb{N}$ is a polynomial and L is decidable by a NDTM N that runs in time $p(n)$. For every $x \in L$, there is a sequence of nondeterministic choices that makes N reach q_{accept} on input x. We can use this sequence as a certificate for x. This certificate has length $p(|x|)$ and can be verified in polynomial time by a deterministic machine.

Conversely, if $L \in$ **NP**, then we describe a polynomial time NDTM N that decides L. On input x, it uses the ability to make nondeterministic choices to write down a string u of length $p(|x|)$. (Having transition δ_0 correspond to writing a 0 and δ_1). Then it runs the deterministic verifier □

2.0.2 Reducibility and NP-Completeness

Definition 2.5. A language $L \subseteq \{0,1\}^*$ is **polynomial-time Karp reducible to a language** $L' \subseteq \{0,1\}^*$ (sometimes shortened to just "polynomial-time reducible"), denoted by $L \leq_p L'$ if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ s.t. for every $x \in \{0,1\}^*$, $x \in L$ iff $f(x) \in L'$

We say that L' is **NP-hard** if $L \leq_p L'$ for every $L \in$ **NP**. We say that L' is **NP-complete** if L' is **NP**-hard and $L' \in$ **NP**

Theorem 2.6. 1. *(Transitivity) If $L \leq_p L'$ and $L' \leq_p L''$ then $L \leq_p L''$*

2. *If language L is **NP**-hard and $L \in$ **P** then **P** $=$ **NP***

3. *If language L is **NP**-complete, then $L \in$ **P** iff **P** $=$ **NP***

Theorem 2.7. *The following language is **NP**-complete*

$$\mathtt{TMSAT} = \{\langle \alpha, x, 1^n, 1^t \rangle : \exists u \in \{0,1\}^n \text{ s.t. } M_\alpha \text{ outputs } 1 \text{ on input } \langle x, u \rangle \text{ within } t \text{ steps}\}$$

Proof. There is a polynomial p and a verifier TM M s.t. $x \in L$ iff there is a string $u \in \{0,1\}^{p(|x|)}$ satisfying $M(x,u) = 1$ and M runs in time $q(n)$ for some polynomial q.

Map every string $x \in \{0,1\}^*$ to the tuple $\langle \llcorner M \lrcorner, x, 1^{p(|x|)}, 1^{q(m)}$ where $m = |x| + p(|x|)$ and $\llcorner M \lrcorner$ denotes the representation of M as string.

$$\begin{aligned}
&\langle \llcorner M \lrcorner, x, 1^{p(|x|)}, 1^{q(m)} \rangle \in \mathtt{TMSAT} \\
&\Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) \text{ outputs 1 within } q(m) \text{ steps} \\
&\Leftrightarrow x \in L
\end{aligned}$$

□

2.0.3 The Cook-Levin Theorem: Computation is Local

We denote by SAT the language of all satisfiable CNF formulae and by 3SAT the language of all satisfiable 3CNF formulae

Theorem 2.8 (Cook-Levin Theorem). *1. SAT is **NP**-complete*

*2. 3SAT is **NP**-complete*

Lemma 2.9 (Universality of AND, OR, NOT). *For every Boolean function $f : \{0,1\}^l \to \{0,1\}$, there is an l-variable CNF formula φ of size $l2^l$ s.t. $\varphi(u) = f(u)$ for every $u \in \{0,1\}^l$, where the size of a CNF formula is defined to be the number of $\wedge/\vee$ symbols it contains*

Proof. For every $v \in \{0,1\}^l$, there exists a clause $C_v(z_1, \dots, z_l)$ s.t. $C_v(v) = 0$ and $C_v(u) = 1$ for every $u \neq v$.

We let φ be the AND of all the clauses C_v for v s.t. $f(v) = 0$

$$\varphi = \bigwedge_{v:f(v)=0} C_v(z_1, \dots, z_l)$$

Note that φ has size at most $l2^l$. □

Lemma 2.10. *SAT is **NP**-hard*

Proof. Let L be an **NP** language. By definition, there is a polynomial time TM M s.t. for every $x \in \{0,1\}^*$, $x \in L \Leftrightarrow M(x,u) = 1$ for some $u \in \{0,1\}^{p(|x|)}$, where $p : \mathbb{N} \to \mathbb{N}$ is some polynomial. We show L is polynomial-time Karp reducible to SAT by describing a polynomial-time transformation $x \to \varphi_x$ from strings to CNF formulae s.t. $x \in L$ iff φ_x is satisfiable. Equivalently

$$\varphi_x \in \texttt{SAT} \quad \text{iff} \quad \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x \circ u) = 1$$

where $\circ$ denotes concatenation

Assume M

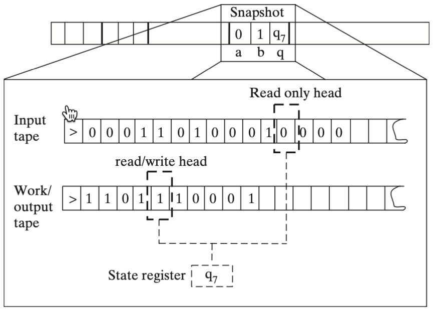
1. M only has two tapes - an input tape and a work/output tape
2. M is an oblivious TM in the sense that its head movement does not depend on the contents of its tapes. That is, M's computation takes the same time for all inputs of size n, and for every i the location of M's head at the ith step depends only on i and the length of the input

We can make these assumptions without loss of generality because for every $T(n)$-time TM M there exists a two-tape oblivious TM $\tilde{M}$ computing the same function in $O(T(n)^2)$. Thus in particular, if $L \in \mathbf{NP}$, then there exists a two-tape oblivious polynomial-time TM M and a polynomial p s.t.

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x \circ u) = 1 \tag{1}$$

Note that because M is oblivious, we can run it on the trivial input $(x, 0^{p(|x|)})$ to determine the precise head position of M during its computation on every other input of the same length.

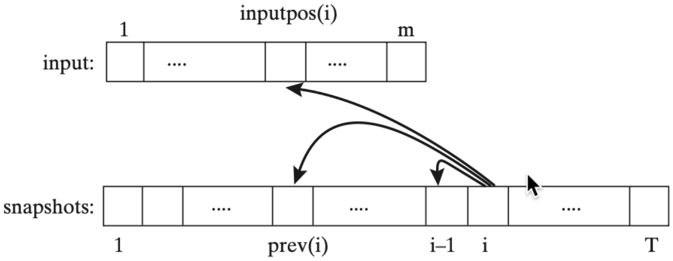
Denote by Q the set of M's possible states and by Γ its alphabet. The **snapshot** of M's execution on some input y at a particular step i is the triple $\langle a, b, q \rangle \in \Gamma \times \Gamma \times Q$ s.t. a, b are the symbols read by M's heads from the two tapes and q is the state M is in at the ith step. Clearly the snapshot can be encoded as a binary string. Let c denote the length of this string, which is some constant depending upon $|Q|$ and $|\Gamma|$



For every $y \in \{0,1\}^*$, the snapshot of M's execution on input y at the ith step depends on its state in the $(i-1)$st step and the contents of the current cells of its input and work tapes.

Suppose somebody were to claim the existence of some u satisfying $M(x \circ u) = 1$ and as evidence, present you with the sequence of snapshots that arise from M's execution on $x \circ u$. How can you tell that the snapshots present a valid computation that was actually performed by M.

Clearly, it suffices to check that for each $i \leq T(n)$, the snapshot z_i is correct given the snapshot for the previous $i-1$ steps. However, since the TM can only read/modify one bit at a time, to check the correctness of z_i it suffices to look at only *two* of the previous snapshots. Specifically, to check z_i we need to only look at the following: z_{i-1}, $y_{\text{inputpos}(i)}$, $z_{\text{prev}(i)}$.



Here y is a shorthand for $x \circ u$. inputpos(i) denotes the location of M's input tape head at the ith step. prev(i) is the last step before i when M's head was in the same cell on its work tape that it is during step i. The reason this small amount of information suffices to check the correctness of z_i is that the contents of the current cell have not been affected between step prev(i) and step i.

Since M is a deterministic TM, for every triple of values to $z_{i-1}, y_{\text{inputpos}(i)}$, $z_{\text{prev}(i)}$, there is at most one value of z_i that is correct. Thus there is some function F that maps $\{0,1\}^{2c+1}$ to $\{0,1\}^c$ s.t. a correct z_i satisfies

$$z_i = F(z_{i-1}, z_{\text{prev}(i)}, y_{\text{inputpos}(i)})$$

Because M is oblivious, the values inputpos(i) and prev(i) do not depend on the particular input i. These indices can be computed in polynomial-time by simulating M on a trivial input.

By (1) , $x \in \{0,1\}^n \in L$ iff $M(x \circ u) = 1$ for some $u \in \{0,1\}^{p(n)}$. The previous discussion shows this latter condition occurs iff there exists a string $y \in \{0,1\}^{n+p(n)}$ and a sequence of strings $z_1, \ldots, z_{T(n)} \in \{0,1\}^c$ (where $T(n)$ is the number of steps M takes on inputs of length $n + p(n)$) satisfying the following four conditions

1. The first n bits of y are equal to x

2. The string z_1 encodes the initial snapshot of M. That is, z_1 encodes the triple $\langle \rhd, \Box, q_{\text{start}} \rangle$.

3. For every $i \in \{2, \ldots, T(n)\}$, $z_i = F(z_{i-1}, z_{\text{prev}(i)}, y_{\text{inputpos}(i)})$.

4. The last string $z_{T(n)}$ encodes a snapshot where the machine halts and outputs 1

The formula φ_x will take variables $y \in \{0,1\}^{n+p(n)}$ and $z \in \{0,1\}^{cT(n)}$ and will verify that y, z satisfy the AND of these four conditions. Thus $x \in L \Leftrightarrow \varphi_x \in \texttt{SAT}$.

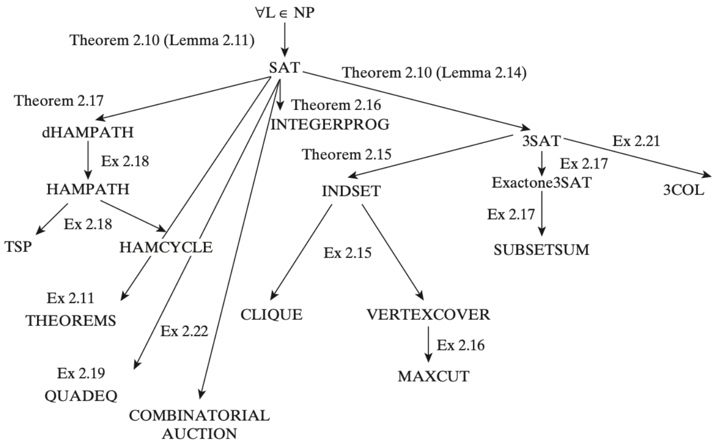
Condition 1 can be expressed as a CNF formula of size $4n$. Conditions 2 and 4 each depend on c variables and hence by Proposition 2.9 can be expressed by CNF formulae of size $c2^c$. Condition 3, which is an AND of $T(n)$ conditions each depending on at most $3c+1$ variables, can be expressedas a CNF formula of size at most $T(n)(3c+1)2^{3c+1}$. Hence the AND of all these conditions can be expressed as a CNF formula of size d(n + T(n)) where d is some constant depending only on M. Moreover, this CNF formula can be computed in time polynomial in the running time of M. □

Lemma 2.11. *SAT* $\leq_p$ *3SAT*

Proof. Suppose φ is a 4CNF. Let C be a clause of φ, say $C = u_1 \vee \bar{u}_2 \vee \bar{u}_3 \vee u_4$. We add a new variable z to the φ and replace C with the pair $C_1 = u_1 \vee \bar{u}_2 \vee z$ and $C_2 = \bar{u}_3 \vee u_4 \vee \bar{z}$. If C is true, then there is an assignment to z that satisfies both C_1 and C_2. If C is false, then no matter what value we assign to z either C_1 or C_2 will be false.

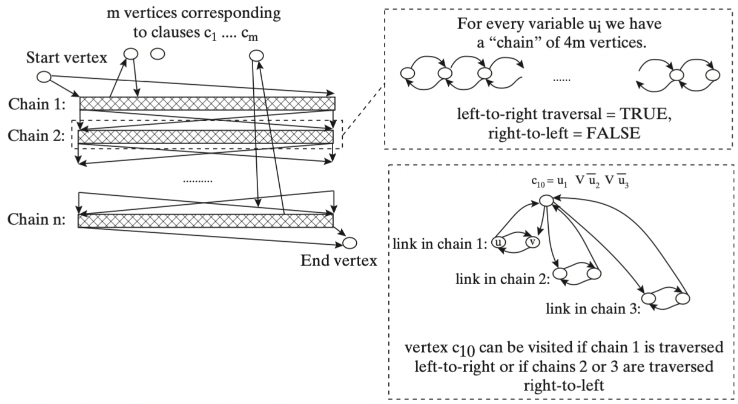
For every clause C of size $k > 3$, we change it into an equivalent pair of clauses C_1 of size $k-1$ and C_2 of size 3. □

2.0.4 The Web of Reductions



A **Hamilton path** in a directed graph is a path that visits all vertices exactly once. Let `dHAMPATH` denote the set of all directed graphs that contain such a path

Theorem 2.12. *dHAMPATH is **NP**-complete*



Proof.
The graph G has

1. m vertices for each of φ's clause $c_1, \dots, c_m$

2. a special starting vertex v_{start} and ending vertex v_{end}

3. n "chains" of $4m$ vertices corresponding to the n variables of φ . A chain is a set of vertices $v_1, \dots, v_{4m}$ s.t. for every $i \in [1, 4m-1]$, v_i and v_{i+1} are connected by two edges in both directions

If C contains the literal u_j, then we take two neighboring vertices v_i, v_{i+1} in the jth chain and put an edge from v_i to C and from C to v_{i+1}. If C contains the literal $\bar{u}_j$ then we construct these edges in the opposite direction. When adding these edges, we never "reuse" a link v_i, v_{i+1} in a particular chain and always keep an unused link between every two used links.

$G \in \texttt{dHAMPATH} \Rightarrow \varphi \in \texttt{SAT}$. Suppose that G has an Hamiltonian path P. We first note that the path P must start in v_{start} and end at v_{end}. Furthermore, we claim that P needs to traverse all the chains in order and, within each each chain, traverse it either in left-to-right order or right-to-left order. □