Lecture 9

Gidon Rosalki

2025-05-25

1 Overview

Theorem 1 (Cook-Levin Theorem)**.**

$$SAT \in NPC$$

Theorem 2. *Both definitions of NP are equivalent.*

2 Cook Levin

Let us recall that

$SAT = \{\varphi :$ There is a CNF formula that satisfies φ, which is to say φ is a boolean formula, constructed from literals, which a

We have shown previously that $SAT \in NP$, $SAT \leq_p 3SAT$, and therefore $SAT \in NPC \implies 3SAT \in NPC$. We need to show that for every $L \in NP$, $L \leq_p SAT$. To do this, we need a function, that translates from w to φ, such that

$$w \in L \Leftrightarrow \varphi \in SAT$$

The problem is we know not what is L. All that we know is that $L \in NP$, which is to say that there is a TM M, non deterministic and polynomial that recognises L.

Be warned, this is a *long* proof, we are going to spend about 1 to 2 hours on it. There are harder proofs that we have seen, and that we will see, but this is long, with a great many parts.

2.1 Configuration reminder

Definition 2.1 (Configuration)**.** *A configuration is a description of the overall state during a run, or formally:*

$$c = uqv : u, v \in \Gamma^* \wedge q \in Q$$

The configuration c describes the general state where the internal state of the machine is q, the contents of the tape is uv, and the head is pointed at the first letter of v.

For example, the initial configuration of the run $M(w)$ is

$$c_0 = q_0 w$$

An accepting configuration is a configuration that contains the state q_{acc}, and a rejection configuration is a configuration that contains the state q_{rej}.

2.2 Sub theorem

Theorem 3. *W.l.o.g. The machine M has a single accepting configuration.*

Proof . We will show how to change M such that it will have a single accepting configuration, and the runtime will remain polynomial. We will change every state q_{acc} to $q_{\text{almost accept}}$, and in this state M will delete the whole tape, and return the head to the beginning of the tape, and move to state q_{acc}. It is clear that there is now only a single accepting state, and additionally we will note that the runtime is extended by at most a polynomial of the length of the input, since the tape that we need to delete contained at most a polynomial number of cells. □

Theorem 4. *W.l.o.g. The machine M only has a single rejecting configuration*

Proof . Similar to the previous proof. □

2.3 Table construction

We will create a run of $M(w)$ by using a table, where every row of the table is matched to a configuration, starting from row 0 representing the configuration c_0. A run will match to a correct filling of the table, and if the final row is an accepting configuration, then the run accepts.
We will build a formula that checks if the filling of the table is correct, and if it finishes in an accepting configuration. So, we have a table, with $t+1$ rows, starting at the initial configuration c_0, and running until c_t, where

$$t = poly(n) : n = |w|$$

We also know that the width of the table, which represents the space used, is also limited in size by t, for similar reasoning. Every cell within the table will contain a letter from $\Gamma \cup Q$. The height of the table t is bound by the runtime, and the width of the table is bound by the length of the longest configuration, which is to say is bound by the space, and is therefore also bound by t.
Filling the table is correct if:

- In row 0 we have the starting configuration
- In row t we have a final configuration, (accepting if the run accepts)
- For all $0 \leq i < t$ it holds that c_i, c_{i+1} are following configurations, which is to say are suitable to a possible step in the run of M. This is to say that c_i, c_{i+1} are equivalent in almost every place, aside from around the location of the head of the machine, and there the differences need to come from the function δ of M

So in the first row, which represents c_0, we expect to see q_0 in the first cell, the letters of the word, and then the rest of the row filled with $\sqcup$. In the bottom row, c_t, we expect to see q_{acc} (if the run accepts), followed by $\sqcup$for the rest of the row.
The immediate location surrounding the head, if the head is at $c_{i,j}$, then it is all the cells in $i, i+1$, and in $j-1, j, j+1$, since those are all the cells that can be changed by the machine, and to which the head can move. The main concept is now understood, so it remains to show:

1. How to formalise all the checks:
   - Of the 0th row
   - Of the tth row
   - Of all the collections of 6 boxes (the immediate surroundings of the head between c_i and c_{i+1}
2. How to convert all the above to CNF literals, and in particular into boolean variables
3. How to create a CNF formula using a TM M' which is deterministic, and polynomial
4. To prove correctness, and runtime

2.4 Building and checking correctness for filling the table

We will call the cells in the table $x_{i,j}$, where $x_{0,0}$ is the upmost, and leftmost cell, and $x_{t,t}$ is the rightmost, bottom cell.

2.4.1 Checking row 0

So we want to check

$$\varphi_{\text{init}}(x_{0,0}, x_{0,1}, \ldots, x_{0,t}) = (x_{0,0} = q_0) \wedge \wedge_{j=1}^{n} (x_{0,j} = w_j) \wedge \wedge_{j=n+1}^{t} (x_{0,j} = \sqcup)$$

We will note that if next every cell is translated to a CNF formula, then φ_{init} will be a CNF formula

2.4.2 checking row t

$$\varphi_{\text{acc}}\left(x_{t,0},\ldots,x_{t,t}\right) = \left(x_{t,0} = q_{\text{acc}}\right) \wedge \wedge_{j=1}^{t}\left(x_{t,j} = \sqcup\right)$$

We want a way to show that 2 continuous rows are suitable to 2 continuous configurations, or rather a correct filling for a group of 6 cells is $\forall a, b, c \in \Gamma$: What are the correct ways to fill the 6 cell grouping when the area undergoes a change? We will look at the function $(q_2, b, R) \in \delta(q_1, a)$

a	b	c
a	b	c

Table 1: 6 collection in table

q_1	a	c
b	q_2	c

Table 2: Possible filling, $\forall c \in \Gamma$

c	q_1	a
c	b	q_2

Table 3: Possible filling $\forall c \in \Gamma$

d	c	q_1
d	c	b

Table 4: Possible filling $\forall c, d \in \Gamma$

where here we have a, and q_2 off the rights of the top and bottom rows, accordingly.

a	c	d
q_2	c	d

Table 5: Possible filling $\forall c, d \in \Gamma$

where we have q_1 and b off the left of the top and bottom rows, accordingly.
We may similarly add checks for all the correct fillings of this group of 6, where the instruction finishes with an order to move to the right.
We will define a check for the 6 cells

$$\varphi_{\text{legal}}^{i,j}\left(x_{i,j}, x_{i,j+1}, x_{i,j+2}, x_{i+1,j}, x_{i+1,j+1}, x_{i+1,j+2}\right)$$

where we check if the filling is correct according to at least one of the options that we described before.

$$\varphi = \varphi_{\text{init}} \wedge \varphi_{\text{acc}} \wedge \wedge_{i=0}^{t} \wedge_{j=1}^{t} \varphi_{\text{legal}}^{i,j}$$

Where the test $\varphi_{\text{legal}}^{i,j}$ returns true **if and only if** The placement of the 6 relevant variables if a placement that is allowed within the placements of $(\Gamma \cup Q)^6$.
We want to translate everything into booleans, which is to say:

- Every variable in the table $x_{i,j}$ will be switched into a set of boolean variables
- Every check we want to formulate into :
  1. A formula using boolean variables
  2. In the style of CNF

For every original variable $x_{i,j}$, we need $\lceil \log_2(|\Gamma| + |Q|) \rceil$ boolean variables. We will use the circumflex symbol to represent the boolean version of the formulas from the previous step:

$$\varphi_{\text{init}}\left(x_{0,0},\ldots,x_{0,t}\right)$$

has only one requirement for each variable, therefore, $\hat{\varphi}_{\text{init}}$ also requires the same thing.
Similarly, we can also describe $\hat{\varphi}_{\text{acc}}$ as a CNF formula. We therefore only need to handle φ_{legal}.
For every placement of 6 original variables, there is a suitable placement for the boolean variables that encode them. For every such row we want the formula to return true if the value for the 6 is correct, and false otherwise. This is to say, we have acquired a truth table of the formula $\hat{\varphi}_{\text{legal}}^{i,j}$, and we want to find a CNF representation that agrees with

this truth table. If we did not need it to be CNF structure, we could solve this as with truth tables in the past, DNF, which is to say, for every row that is true, we create a literal where every variable is connected by $\wedge$, and we connect the literals together with $\vee$.
In order to create a CNF formula, we can create the DNF formula for every row that is **false**, and take the inverse of the resulting formula (DeMorgan), which will give us a CNF formula for $\hat{\varphi}^{i,j}_{\text{legal}}$.
So:

$$\hat{\varphi} = \hat{\varphi}_{\text{init}} \wedge \hat{\varphi}_{\text{acc}} \wedge \wedge^{t}_{i,j=0} \hat{\varphi}^{i,j}_{\text{legal}}$$

With that we have finished to show that there exists a formula $\hat{\varphi}$ which is a CNF formula, and that it is satisfiable **if and only if** $w \in L$. Good job Cook and Levin for creating this, and good job to us for understanding it. We now need to show that there exists a reduction $L \leq_p SAT$, which is to say that there exists a TM M, that is both deterministic, and polynomial, that for a given input w, will return $\hat{\varphi}$, such that

$$w \in L \Leftrightarrow \hat{\varphi} \in SAT$$

which is to say, to show how we create $\hat{\varphi}$ from w by a TM M which deterministic, and polynomial.

1. M' will compute t (some polynomial on $n = |w|$). This is easy in polynomial time
2. M' will create $\hat{\varphi} \wedge \hat{\varphi}$, both are easy in polynomial time
3. M' will run a double loop (on i, j from 0 to t), and for each i, j will print a copy of $\hat{\varphi}^{i,j}_{\text{legal}}$. We will note that all the formulas of $\hat{\varphi}^{i,j}_{\text{legal}}$ are exactly the same, beyond a change in the names of the variables. This is to say that the formula $\hat{\varphi}_{\text{legal}}$ can be part of the encoding of the machine M', and M' only needs to print this encoding again and again with the update i, j, and between every 2 copies, the symbol $\wedge$.

Note, the polynomial that will be used to calculate t from $n = |w|$ exists, and therefore can be encoded within M'.

2.5 Runtime

M' needs to print $\hat{\varphi}_{\text{init}}, \hat{\varphi}_{\text{acc}}$, both in polynomial time. Additionally, M' needs to print for each value $0 \leq i, j < t$ $\hat{\varphi}^{i,j}_{\text{legal}}$. This is t^2 times, which is a polynomial number of times. Overall, we thus have a polynomial runtime

2.6 Correctness

If $w \in L$, then there exists an accepting run fo $M(w)$, therefore there exists a correct way to fill the original table where in the first row c_0 is the starting configuration, in the final row is the accepting configuration, and every two subsequent rows are matched to subsequent configurations in the run. Therefore, every check will pass. Therefore, there exists a placement o boolean variables such that every CNF check passes, which is to say that is a satisfying placement for φ.
If $w \notin L$, then every way to fill the original table is either suitable to a legal run that finishes in a configuration that rejects, or is not suitable to a legal run. In every case, at least one check of $\varphi_{\text{init}}, \varphi_{\text{acc}}$ or $\varphi^{i,j}_{\text{legal}}$ fails. Therefore, at least one CNF formula of these checks of boolean variables returns false, and therefore every placement does not satisfy $\hat{\varphi}$. This is to say $\hat{\varphi} \notin SAT$.

3 Different definitions of NP

We saw 2 definitions for NP:

1. NP: All the languages L which are recognised by a non deterministic, polynomial TM M
2. NP': All the languages L such that there is a polynomial, deterministic verifier M'

3.1 Reminder

A verifier M' for a language L is a machine that accepts w, c and enables the following: If $w \in L$, then there exists c such that $M(w, c) = q_{\text{acc}}$, and if $w \notin L$, then for all c, $M(w, c) = q_{\text{rej}}$. Additionally, for every c, $M(w, c)$ runs in polynomial time by the length of $|w|$.

3.2 Theorem

We want to prove that $NP = NP'$. This is useful because it is easier to show that something is in NP by creating a verifier, than to create an NTM, and if we prove once that they are equivalent, then we can use that forevermore.

$NP' \subseteq NP$: We are given a language L, which has a deterministic polynomial verifier, which is to say a machine M' suitable to the above definition. We want, using M', to build a machine nil M, which is polynomial, and none deterministic, that recognises L. W.l.o.g. The length $|c|$ is bound by a polynomial of $|w|$. Explanation: M' runs in polynomial time of $|w|$, and therefore cannot read more than a polynomial number of letters from c.

M will run as follows: Create a string c', of length polynomial in $|w|$, in a non deterministic manner. Then run like M' on w, c.
Correctness: If $w \in L$, then there is c such that $M'(w, c) = q_{\text{acc}}$, therefore there is an assumption of M for the letter values of c that will cause M to return q_{acc}. IF $w \notin L$, then for every c, $M'(w, c) = q_{\text{rej}}$, and therefore for all c that M will create, the run will end in q_{rej}.
Runtime: Create c is bound by the length of c, which is to say wlog is bound from above by a polynomial in $|w|$. Additionally, the step of running $M'(w, c)$ is known to take time that is polynomial in $|w|$. In total, we have a polynomial runtime in $|w|$.

$NP \subseteq NP'$: Given a language L, which has an MT M, that is polynomial, and non deterministic. We want to use it to build a TM M' which is a polynomial verifier for L. We need to create a function

$$\delta : Q \times \Gamma \to P\left[Q \times \Gamma \times \{L, R\}\right]$$

The number of options k for choosing non deterministically in every step of the run is at most $|Q| \cdot |\Gamma| \cdot 2$.
We will look at the run of $M(w)$. In every step of the run there is a non deterministic choice made by the function δ. This defines a string in which every letter is a number between 1 and k. The length of the string is the number of steps of the run, which is to say, polynomial in $|w|$. We can define a TM M', which is deterministic, which accepts the input w, and a string such as this, called c, which will simulate $M(w)$, where taking a non deterministic step, is a step chosen according to a letter in c. That is to say M' runs like M, at every step similar to M, reads an additional letter from c, and uses that letter to choose which step of δ to run. We will note that M' is deterministic.
Correctness: If $w \in L$, then there exists an accepting run of $M(w)$, which is to say that there exists a series of non deterministic choices of δ that cause M to arrive at q_{acc}, and therefore if c is the string that describes this order of choices, then $M'(w, c)$ will return q_{acc}.
If $w \notin L$, then every run of $M(w) = q_{\text{rej}}$, therefore for every string c, it holds that $M'(w, c) = q_{\text{rej}}$.
Runtime: The length of c is polynomial in $|w|$. The number of steps of the simulation that M' does is the number of steps of $M(w)$, which is to say polynomial in the length of w. Simulation of each step requires at most the length of the tape, which is to say at most, polynomial in $|w|$. In total, runtime is polynomial in $|w|$.