

# 1

Let  $SET - SPLITTING = \{ \langle S, C \rangle \}$  where  $S$  is a finite set of  $n$  elements and  $C = \{C_1, \dots, C_k\}$  is a collection of  $k$  subsets of  $S$ , and we can color the elements of  $S$  red or blue so that no subset  $C_i \in C$  contains only elements of one color. Prove that  $SET - SPLITTING$  is NP-complete.

**Prove  $SET - SPLITTING \in NP$**

To prove  $SET - SPLITTING \in NP$  we need to show  $\exists V \in P$  that verifies a string  $w \in SET - SPLITTING$  given the non-deterministic guess of  $SET - SPLITTING$ .

If we are given a guess of color assignments for each element, we need to verify each clause has at least one of each color. We iterate through each clause checking if there is at least one red and one blue element. If a clause contains only one color we reject. If we reach the end we accept. This is polynomial time because we iterate through each clause and for each element in the clause we find its correct color. So this is  $O(n^2)$  so  $SET - SPLITTING \in NP$  because  $\exists V \in P$ .

**Prove  $3 - SAT \leq_p SET - SPLITTING$**

**Idea**

Create a function  $f$  that maps  $F$  from  $3 - SAT$  to  $\langle S, C \rangle$  for  $SET - SPLITTING$  and  $F \in 3 - SAT \leftrightarrow f(F) \in SET - SPLITTING$ .

$F$  has  $n$  variables,  $f$  creates an  $S$  with  $2n$  variables where a variable  $x_i \in F$  is mapped to  $s_i$  and  $\neg x_i \in F$  is mapped to  $s_{i+n}$ . We can ensure this by creating the first  $n$  collections where for each variable  $x_i \in F$  create  $C_i = (s_i, s_{i+n})$ . This ensures that the negation works correctly for each variable  $x_i \in F$ . Now we translate each clause from  $F$  to a new collection in  $C$ . That is, the  $i$ th clause in  $F$  maps to the  $C_{i+n}$  clause. Each variable in  $c_i$  is added to  $C_{i+n}$  unless a given  $x_j$  is negated in which case we would add  $s_{j+n}$  to  $C_{i+n}$  instead of  $s_j$ .

**Proof**

$F \in 3 - SAT \rightarrow f(F) \in SET - SPLITTING$

$F \in 3 - SAT$  means there exists an assignment for each variable such that at least one literal in each clause is true. If we take that assignment we can translate that to  $f(F)$  where if the  $i$ th variable in  $F$  is true then the  $i$ th variable in  $f(F)$  is blue and red otherwise. This makes the  $i + n$ th variables the opposite colors of the  $i$ th variable because of the first  $n$  clauses in  $f(F)$ . Every other clause matches exactly to the clauses in  $F$  and we know each clause in  $F$  must have at least one  $T$

$$F \notin 3 - SAT \rightarrow f(F) \notin SET - SPLITTING$$

$$f(F) \in SET - SPLITTING \rightarrow F \in 3 - SAT$$

$$f(F) \notin SET - SPLITTING \rightarrow F \notin 3 - SAT$$

Therefore  $\exists f$  such that  $F \in 3 - SAT \leftrightarrow f(F) \in SET - SPLITTING$  then  $3 - SAT \leq_p SET - SPLITTING$  and  $SET - SPLITTING \in NP$  so  $SET - SPLITTING$  is NP-complete.

## 2

A ladder is a sequence of strings  $s_1, s_2, \dots, s_k$ . where every string differs from the preceding one in exactly one character. For example the following is a ladder of English words, starting with “head” and ending with “free”: head, hear, near, bear, beer, deer, deed, feed, feet, fret, free. Let  $LADDERDFA = \{\langle M, s, t \rangle \mid M \text{ is a DFA and } L(M) \text{ contains a ladder of strings, starting with } s \text{ and ending with } t\}$ . Show that  $LADDERDFA$  is in PSPACE.

**Prove**  $LADDERDFA \in PSPACE$

**Idea**

Create machine  $M_{LADDER}$  which does the following:

Take the  $M$ , start, and end as input.

Count size of start (count variable is  $O(\log n)$  space) and let this be  $n$

Initialize a count variable  $m$  and enumerate in lexical graphical order all strings of size  $n$  and for each string run  $M$  on it and increment  $m$  by 1

if  $M$  accepts. Enumerating is  $O(n)$  space (we only have one string at a time then overwrite it with the next string). Count variable is  $O(\log n)$  if we use binary. Running  $M$  on a string is also  $O(n)$  because  $M$  is a DFA and only reads the string and accepts / rejects at end of string.

Thus the above step is all  $O(n)$  space.

We can then call a helper method  $M'(M, \text{start}, \text{end}, 0, m)$  and accepts if  $M'$  accepts rejects otherwise.

Create machine  $M'$  which does the following:

Take  $M$ , start, end, iteration, max as input.

Reject if iteration  $>$  max ( $O(n)$  space)

Reject if  $M$  rejects start or end ( $O(n)$  space)

Accept if start = end ( $O(n)$  space)

Copy start onto working tape ( $O(n)$  space)

for  $i$  in range |start|: ( $O(\log n)$  space):

    for  $j$  in  $\Sigma$ : ( $O(1)$  space)

        Change the  $i$ th character of start to be  $j$  ( $O(n)$  space)

        Compare new string with end and accept if they are the same

Otherwise  $\exists m \forall a, b [(a = \text{start} \wedge b = m) \vee (a = m \wedge b = \text{end})] \rightarrow (M'(a, b, \text{iteration} + 1, \text{max}/2))$ .

In other words, see if start and end are one away, reject if start or end are not in  $L(M)$  and reject if we have exceeded the number of strings in  $L(M)$  or size  $n$ . Then if none of those conditions are true we see if there is a middle element that can connect start to end.

## Proof

Space complexity of  $M_{LADDER}$  is  $O(\log n) + O(n) + O(\log n) +$  complexity of  $M'$ . This means space complexity of  $M_{LADDER}$  is  $O(n) +$  space complexity of  $M'$ . Therefore  $M_{LADDER} \in PSPACE$  iff  $M' \in PSPACE$ .

Space complexity of  $M'$  before recursion is  $O(n) + O(n) + O(n) + O(n) + O(\log n) + O(1) + O(n) + (*O(n)) = O(n)$ . The  $*O(n)$  is space for the quantifiers. So space

complexity of  $M'$  is  $O(n * d)$  where  $d$  is the depth of the stack. So we need to prove that  $d \in O(n^k)$ .

At each level of recursion, we split the ladder in half so this would be an upper bound (for space complexity) of  $2^d = m$  where  $m$  is the number of strings of size  $n$  in  $L(M)$ . Which means that  $m$  has an upper bound of  $|\Sigma|^n$ . Therefore we have  $2^d = |\Sigma|^n$ . If we take  $\log_2$  of both sides we get  $d = cn$  where  $c$  is some constant ( $2^c = |\Sigma|$ ). Thus  $d \in O(n)$ .

Therefore the total space complexity of  $M' \in O(n^2)$  so  $M_{LADDER} \in O(n^2)$  so  $LADDERDFA \in PSPACE$

### 3

Define  $ALBA = \{\langle M, w \rangle | M \text{ is a linear bounded Turing machine that accepts } w\}$ . A linear bounded Turing machine is one where the head cannot move off of the cells that store the input. Any move to the right of the input (or the left off the tape) will result in the head staying in the same cell. Prove that  $ALBA$  is PSPACE-complete

**Prove**  $ALBA \in PSPACE$

Create machine  $M_A$  that decides  $ALBA$ .

$M_A$  runs  $M$  on  $w$  and accepts if  $M$  accepts  $w$  otherwise rejects.

Since  $M$  is a linear bound TM it uses  $|w|$  space so  $M_A$  only uses  $|w|$  space. Therefore  $ALBA \in O(n) \in P$

**Prove**  $\forall L \in PSPACE, L \leq_p ALBA$

**Idea**

Let  $M'$  take input  $x$  as input and  $M$  decides  $L$  in  $O(n^k)$  space. Create a function  $f$  for  $M'$  that takes  $x$  as the input and outputs  $\langle M_{LB}, w \rangle$  where  $w = x\_|x|^{k+1}$ . In otherwords  $f$  outputs a function that decides  $L$  in  $O(n^k)$  space and a new string  $w$  which is  $x$  followed by  $n^{k+1}$  blanks.  $M_{LB}$  is  $M$  but is bound to the input size, it still shares the same transition function but there is not an infinite amount of blanks after the input.

## Proof

$$x \in L \rightarrow f(x) \in ALBA$$

If  $x \in L \wedge L \in PSPACE \exists M$  that accepts  $x$  in  $O(n^k)$  space. Since we know that the machine uses at  $O(n^k)$  space, the machine will not run into an error if we limit it to  $O(n^{k+1})$  space. We can translate  $M$  to  $M_{LB}$  which is now bound to the input size, and we increase the input size to  $O(n^{k+1})$ .  $M_{LB}$  will accept the input so  $\langle M_{LB}, w \rangle \in ALBA$ . Meaning  $x \in L \rightarrow f(x) \in ALBA$  is true.

$$x \notin L \rightarrow f(x) \notin ALBA$$

If  $x \notin L \wedge L \in PSPACE \exists M$  that rejects  $x$  in  $O(n^k)$  space. Since we know that the machine uses at  $O(n^k)$  space, the machine will not run into an error if we limit it to  $O(n^{k+1})$  space. We can translate  $M$  to  $M_{LB}$  which is now bound to the input size, and we increase the input size to  $O(n^{k+1})$ .  $M_{LB}$  will reject the input so  $\langle M_{LB}, w \rangle \notin ALBA$ . Meaning  $x \notin L \rightarrow f(x) \notin ALBA$  is true.

$$f(x) \in ALBA \rightarrow x \in L$$

If  $f(x) \in ALBA$  then our machine  $M_{LB}$  accepts on input  $w$ .  $M_{LB}$  accepting  $w$  is synonymous to a machine  $M$  that decides  $L$  accepting  $x$  with a limit of  $O(n^{k+1})$  space (which is valid because  $M$  uses  $O(n^k)$  space). Therefore  $f(x) \in ALBA \rightarrow x \in L$  is true.

$$f(x) \notin ALBA \rightarrow x \notin L$$

If  $f(x) \notin ALBA$  then our machine  $M_{LB}$  rejects on input  $w$ .  $M_{LB}$  rejecting  $w$  is synonymous to a machine  $M$  that decides  $L$  rejecting  $x$  with a limit of  $O(n^{k+1})$  space (which is valid because  $M$  uses  $O(n^k)$  space). Therefore  $f(x) \notin ALBA \rightarrow x \notin L$  is true.

Therefore  $\forall L \in PSPACE \exists f$  such that  $x \in L \leftrightarrow f(x) \in ALBA$  then  $L \leq_p ALBA$  and  $ALBA \in PSPACE$  so  $ALBA$  is PSPACE-complete.