Exercises on space complexity

Exercise 1: Let A be an algorithm of space complexity $s(n)$.

Show that there is an algorithm A' such that

- $\mathcal{L}(A) = \mathcal{L}(A')$
- A' has space complexity $s'(n) = O(s(n) + \log n)$
- A' does not scan the input-tape beyond the boundaries of the input

Proof: we proceed in two steps

1) We prove that on input x, there is an algorithm A_1 such that
   - $\mathcal{L}(A_1) = \mathcal{L}(A)$
   - A_1 does not scan the input-tape beyond location $2^{O(s(n) + \log_2 n)}$ from the input

This proof is analogous to the one that we did in class to show that a poly-space bounded (N)TM is equivalent to one that has running time $t(n) \leq c^{q(n)}$ with $q(n) = O(s(n))$ (where $s(n)$ is a polynomial space bound)

We showed $q(n) = 2 \cdot s(n) + d$, where $c = |\Gamma| + |Q|$

In our case:

- $c^{q(n)} = 2^{\log_2 c \cdot q(n)} = 2^{O(s(n))}$
- we have also the position on the input tape that contributes to the configuration:

$$\Rightarrow n \cdot 2^{O(s(n))} = 2^{\log_2 n} \cdot 2^{O(s(n))} = 2^{O(s(n) + \log_2 n)}$$

different configurations at most

Note: a TM with running time $t(n) \leq 2^{O(s(n) + \log n)}$ can scan at most $2^{O(s(n) + \log n)}$ cells of the input tape

2) We modify the algorithm A_1 in such a way that it does not move beyond the input.

The resulting algorithm A' works as follows:

- whenever A_1 would move right past the end of the input, A' instead:
  - does not move past the end of the input, but maintains a counter on the work tape
  - whenever A_1 moves right, the counter is incremented
    —"— left, —"— decremented

  In this way, A' can keep track of the position of the input head of A_1

  Whenever A_1 moves back again over the input symbol, A' does not update the counter (leaving it to 0)

- A' operates similarly whenever A_1 moves left past the beginning of the input

How much space does the counter use:

Since A_1 does not scan the input tape beyond $s_I(n) = 2^{O(s(n)+\log n)}$ the counter takes $\log_2 s_I(n) = O(s(n) + \log n)$

Hence, the total space used by A' is

$$s(n) + O(s(n) + \log 2) = O(s(n) + \log n)$$

Exercise 2: Let A be an algorithm of space complexity s. E 11.3

Show that there is an algorithm A' such that

- A' computes the same function as A, i.e. $A'(w) = A(w)$ $\forall w \in \{0,1\}^*$
- A' has space complexity $s'(n) = s(n) + O(\log l(n))$

  where $l(n) = \max_{w \in \{0,1\}^n} |A(w)|$ is the size of the

  maximum output for input x of length n
- A' never rewrites on the same location of its output tape

Proof:

A' proceeds in successive iterations each time simulating the whole computation of A:

in the i-th iteration, A' outputs the i-th bit of $A(x)$

When emulating A, in its i-th iteration A' proceeds as follows:

- it does not directly (re)write on the output tape
- instead, it maintains on the work tape:
  - the counter i of the next output bit that will be written
  - a counter c of the bit that A is currently writing
  - the value of the bit written by A in position i
- when A would write an output bit, A' operates depending on the values of i and c:
  - if $i \neq c$, then A' does not output anything
  - if $i = c$, then A' stores the written bit on its worktape
- At the end of its simulation, A' outputs the stored bit to the i-th position of the output tape

How much space $s'(n)$ does A' use on the working tape for inputs of length n?

- $s(n)$ cells, since it performs the computation of A
- the space for the counters i_e and c_o
  - i_e and c_o have to count positions on the output tape, and hence will use $\log_2 \ell(n)$ bits each.

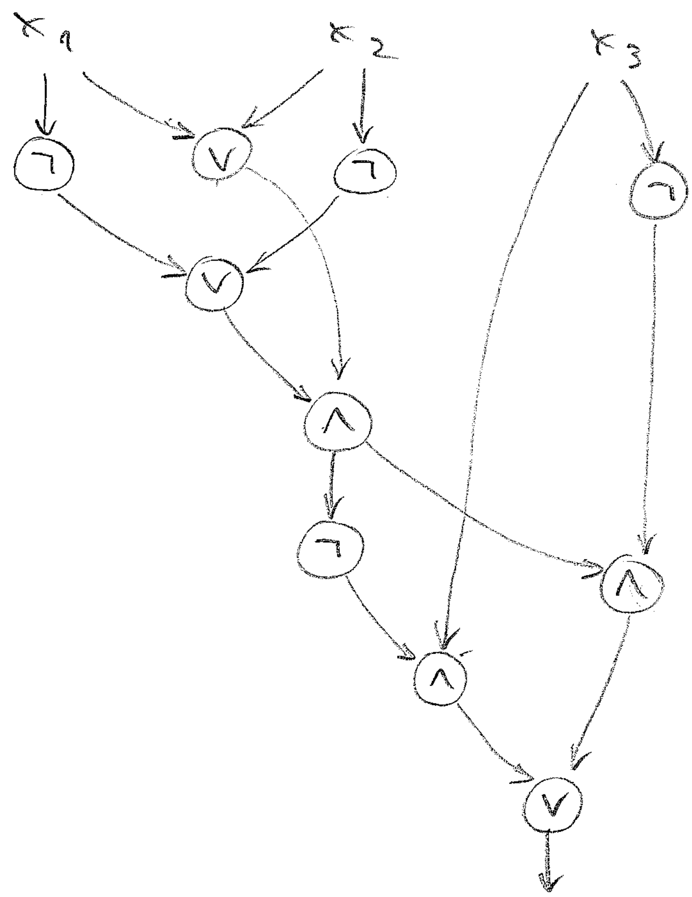
We get that $s'(n) = s(n) + O(\log_2 \ell(n))$

Exercises on boolean circuits

Exercise 1: Consider the boolean expression

$$E = (x_3 \wedge \neg((x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))) \vee (\neg x_3 \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$$

Construct a boolean circuit that computes the value of E, given inputs for x_1, x_2, x_3



What is the size of this circuit? 6 ($\neg$ does not count)
What is the depth — " — ? 4

How does the size compare to the length of E

Exercise: Reduction from Reachability to Circuit Value

Reachability: given a directed graph $G = (V, E)$
with $V = \{1, \dots, n\}$ and $E \subseteq V \times V$,
is there a path from node 1 to node n in G.

Circuit Value: given a Boolean circuit C without input variables, is the output of C equal to T?

We show a logspace reduction of Reachability to Circuit Value, i.e. we show how to construct in logspace from a directed graph G a circuit $R(G)$ such that:

1 is reachable from n in G iff
the value of $R(G)$ is T.

Notice that the key point is to compute $R(G)$ in logspace.
In $R(G)$ we use gates of two forms:

1) g_{ijk}, with $1 \leq i, j \leq n$ and $0 \leq k \leq n$

Intuitively, g_{ijk} is true iff $\; (i) \to \underbrace{\bigcirc \to \bigcirc \to \cdots \bigcirc}_{\text{all} \leq k} \to (j)$

i.e., there is a path from i to j not using any intermediate node bigger than k

2) h_{ijk}, with $1 \leq i, j, k \leq n$ iff

$$(i) \to \underbrace{\bigcirc \cdots \to \bigcirc \to (k) \to \bigcirc \cdots \bigcirc}_{\text{all} \leq k} \to (j)$$

i.e., there is a path from i to j not using any intermediate node bigger than k, but using k as intermediate node

We describe now the gates and how they are connected.

- for $k=0$, all g_{ij0} gates are constant gates

  $g_{ij0} = T$ iff - $i=j$, or
  - $i \to j$ in G

  } This is how G is reflected in $R(G)$

  (note that there are no h_{ij0} gates)

- for $k = 1, 2, \ldots, n$
  - h_{ijk} is an AND gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$
  - g_{ijk} is an OR gate with predecessors $g_{ij,k-1}$ and h_{ijk}

The output gate is $g_{1n,n}$

Note that $R(G)$ can be computed from G in logarithmic space.

Note that the circuit $R(G)$ is legitimate, since it contains no cycles: we can rename the gates $1, 2, \ldots, 2n^3+n^2$,
- in non-decreasing order of the third index, and
- with h_{ijk} preceding g_{ijk}

We have to show that the value of the output gate of $R(G)$ is T iff there is a path from 1 to n in G.

We prove by induction on k that the values of the gates correspond to the informal meaning we gave them:
- for $k=0$: this holds
- if it is true up to $k-1$, the definitions of g_{ijk} and h_{ijk} guarantee that it is true also for k. □

Exercise: A boolean function f is said to be monotone if it has the following property: if one of the values changes from 0 to 1, then the value of f does not change from 1 to 0.

We show that f is monotone iff it can be expressed by a circuit with only AND and OR gates.

"$\Leftarrow$" Consider a circuit C with only AND and OR gates expressing f.

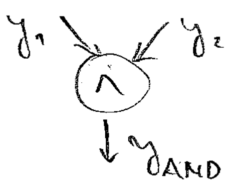
We show by induction on the depth of a node N:
- if the value of an input x changes from 0 to 1 then the value of N does not change from 1 to 0.

Base: depth$(N) = 0$, then N is either an input or a constant node.
- if it is a constant, its value does not change
- if it is an input different from x, — " —
- if it is input x, its value changes from 0 to 1 (and not from 1 to 0)

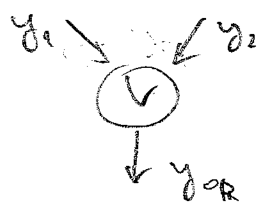
Induction: suppose that for all nodes at level k, the value does not change from 1 to 0.

Consider a node N at level $k+1$. We show that the value of N does not change from 1 to 0.

Case 1: N is an AND node



Case 2: N is an OR node



We need to consider various subcases, corresponding to the changes of y_1, y_2 from 0 to 1.

old		new		y_{AND}		y_{OR}	
y_1	y_2	y_1	y_2	old	new	old	new
0	0	1	1	0	1	0	1
0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	1
0	1	1	1	0	0	1	1
1	0	1	1	0	0	1	1

We see that both for an AND node and for an OR node the output value never changes from 1 to 0.

"$\Rightarrow$" We show by induction on n that every monotone boolean function $f(x_1, \dots, x_n)$ of n variables can be represented by a circuit with AND and OR gates only.

Base case: 0 arguments: f is constant, and hence monotone

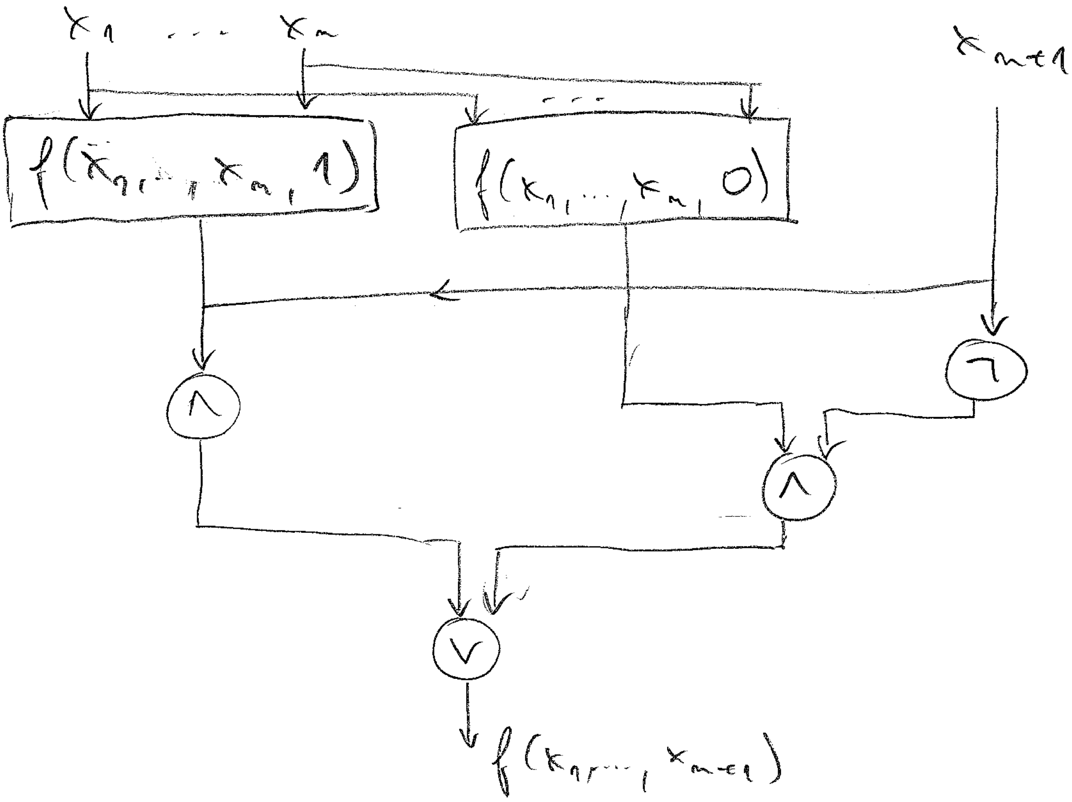
Inductive case: assume the claim holds for n.

We show it holds for a function $f(x_1, \dots, x_{n+1})$.

We exploit the fact that

$$f(x_1, \dots, x_n, x_{n+1}) = (x_{n+1} \wedge f(x_1, \dots, x_n, 1)) \vee (\overline{x}_{n+1} \wedge f(x_1, \dots, x_n, 0))$$

Hence, we can construct a circuit C_f computing $f(x_1, \dots, x_n, x_{n+1})$ as follows:



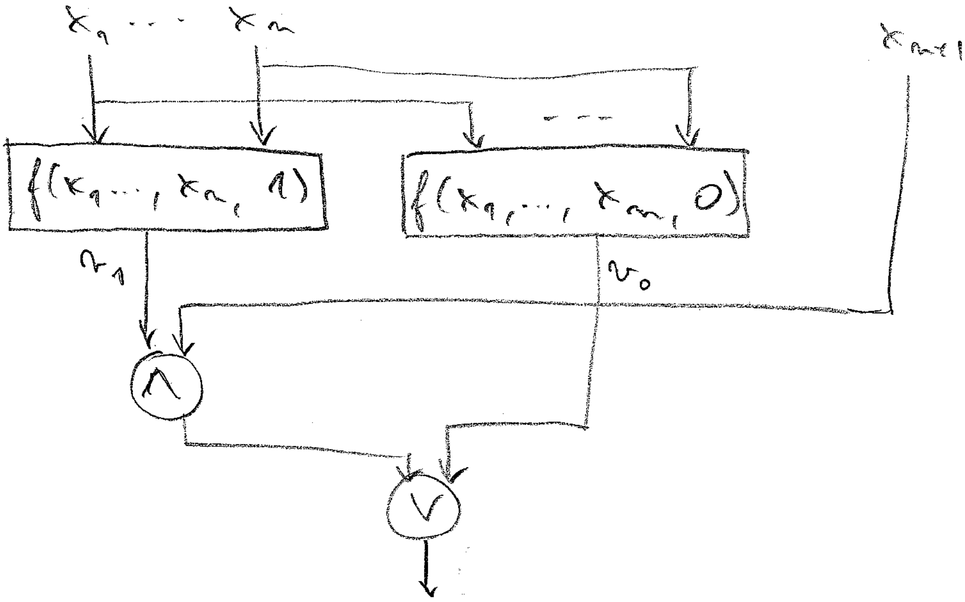
Observe that, since $f(x_1, \dots, x_{n+1})$ is monotone, we have that also $f(x_1, \dots, x_n, 0)$ and $f(x_1, \dots, x_n, 1)$ are monotone.

Hence, since $f(x_1, \dots, x_n, 0)$ and $f(x_1, \dots, x_n, 1)$ are n-variable monotone functions, by inductive hypothesis they can be represented by circuits with AND and OR gates only.

Hence, it suffices to show that we can get rid of the only remaining NOT gate.

Consider the following circuit C_f' in which we have eliminated the NOT gate.

E 11.11



Let us consider the possible values of f in C_f and C_f':
It depends on the values of x_n, v_0, v_1

	x_n	v_0	v_1	C_f	C_f'
1)	0	0	0	0	0
2)	0	0	1	0	0
3)	0	1	0	1	1
4)	0	1	1	1	1
5)	1	0	0	0	0
6)	1	0	1	1	1
7)	1	1	0	0	1
8)	1	1	1	1	1

Note that $C_f = C_f'$, except for case (7).

However, since $f(x_1, \ldots, x_{n+1})$ is monotone, cases (3) and (7) cannot occur, since they would mean that $f(x_1, \ldots, x_n, x_{n+1})$ changes from $v_0 = 1$ for $x_{n+1} = 0$ to $v_1 = 0$ for $x_{n+1} = 1$.

Hence, $C_{f'}$ is the correct circuit consisting of AND and OR gates only and computing $f(x_1, \ldots, x_{n+1})$.