Introduction to Complexity Theory*
Lecture 22: Monotone Circuit Depth and Communication Complexity

Guest Lecturer: Ran Raz
Notes taken by Yael Tauman and Yoav Rodeh

June 21, 1999

Summary: One of the main goals of studying circuit complexity is to prove lower bounds on the size and depth of circuits computing specific functions. Since studying the general model gave few results, we will concentrate on monotone circuits. The main result is a tight nontrivial bound on the monotone circuit depth of *st-Connectivity*. This is proved via a series of reductions, the first of which is of significant importance: A connection between circuit depth and communication complexity. We then get a communication game and proceed to reduce into other such games, until reaching the game $FORK$, and the conclusion that proving a lower bound on its communication complexity will give a matching lower bound on the monotone circuit depth of *st-Connectivity*.

1 Introduction

Turing machines are abstract models used to capture our concept of computation. However, we tend to miss some complexity properties of functions when examining them from the Turing machine point of view. One such central property of a function, is how efficiently it can be run in parallel. This property is best observed when we use the circuit model — by the depth of the circuit realizing the function. Another motivation for prefering the circuit model to the Turing machine model, is the hope that using advanced combinatorial methods will more easily give lower bounds to the size of circuits, and hence to the running time of Turing machines.

Recall that we need only examine circuits made up of $NOT(\neg)$ $OR(\vee)$ and $AND(\wedge)$ gates, any other gate can be simulated with constant blowup in the size and depth of the circuit. We may also assume all the NOT gates are at the leaf level because using De-Morgan rewrite rules, we do not increase the depth of the circuit at all, and may increase its size by a constant factor of 2 at most. In this lecture we will only discuss bounded fan-in circuits, and therefore may assume all gates to be of fan-in 2 (except NOT).

As always, our goal is to find (or at least prove the existence of) hard functions. In the context of circuit complexity we measure hardness by two parameters:

Definition 1 (Depth, Size): *Given* $f : \{0,1\}^n \rightarrow \{0,1\}$, *we define:*

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

1. *Depth*$(f) \stackrel{\text{def}}{=}$ *The minimum depth of a circuit computing* f*, where the depth of a circuit is the maximum distance from an input leaf to the output (when the circuit is viewed as a directed acyclic graph).*

2. *Size*$(f) \stackrel{\text{def}}{=}$ *The minimum size of a circuit computing* f*, where the size of a circuit is the number of gates it contains.*

Note that these quantities do not necessarily correlate: A circuit that computes f and has size $Size(f)$ and depth $Depth(f)$ may not exist. In other words, it is possible that every circuit of minimal size does not achieve minimal depth.

We will first prove the existence of hard functions:

1.1 Hard Functions Exist

There are no explicit families of functions that are proved to need large size circuits, but using counting arguments we can easily prove the existence of functions with size that is exponential in the size of their input.

Proposition 1.1 *For large enough* n*, there exists a function* $f : \{0,1\}^n \to \{0,1\}$ *s.t.* $Size(f) > \frac{2^n}{n^2}$.

Proof: First easy observation is that the number of functions $\{f \mid f : \{0,1\}^n \to \{0,1\}\}$ is exactly 2^{2^n}, since each such function can be represented as a $\{0,1\}$ vector of length 2^n.

We will now upper bound the number of circuits of size s. The way we approach the problem is by adding one gate at a time, starting from the inputs. At first we have $2n$ inputs — the variables and their negations. Each gate we add, is either an OR or an AND gate, and its two inputs can be chosen from any of the original inputs or from the outputs of the gates we already have. Therefore, for the first gate we have $\binom{2n}{2}$ choices for the the inputs and another choice between OR and AND. For the second gate we have exactly the same, except now the number of inputs to choose from is increased by one. Thus, the number of circuits of size s is bounded by:

$$\prod_{i=0}^{s-1} 2 \cdot \binom{2n+i}{2} < \prod_{i=0}^{s-1} 2 \cdot \frac{(2n+i)^2}{2} < \prod_{i=0}^{s-1} (2n+s)^2 = (2n+s)^{2s} = 2^{2s \cdot \log(2n+s)}$$

We wish to prove that the number of circuits of size $s = \frac{2^n}{n^2}$ is strictly less than the number of functions on n variables, and hence prove that there are functions that need circuits of size larger than s. For this we need to prove:

$$2^{2s\log(2n+s)} < 2^{2^n}$$
$$\Updownarrow$$
$$2s\log(2n+s) < 2^n$$

Which is obviously true for $s < \frac{2^n}{n^2}$, since for large enough n:

$$2s\log(2n+s) < 2 \cdot \frac{2^n}{n^2}\log(2^n) = 2^n \cdot (\frac{2}{n}) < 2^n$$

■

If we examine the proof carefully, we can see that actually most functions need a large circuit. Thus it would seem that it should be easy to find such a hard function. However, to the shame of all involved, the best known lower bounds for size and depth of "explicitly given" functions (actually families of functions) are:

$$\begin{aligned} Size &\geq 4n \\ Depth &\geq 3\log(n) \end{aligned}$$

We therefore focus on weaker models of computation:

1.2 Bounded Depth Circuits

The first model we consider is that of bounded depth circuits. There are two deviations from the standard model. The first is that we artificially bound the depth of the circuit, and only consider the size of the circuit as a parameter for complexity. This immediately implies the other difference from the standard model: We do not bound the fan-in of gates. This is because otherwise, if we bound the depth to be a constant d, we automatically bound the size to be less than 2^d which is also a constant. This makes the model uninteresting, therefore we allow unbounded fan-in. Notice that any function can be computed by a depth 2 circuit (not counting NOT's) by transforming the function's truth table into an OR of many AND's. However, this construction gives exponential size circuits. Several results were reached for this model (see Lecture 20), but we will focus on a different model in this lecture.

2 Monotone Circuits

Monotone circuits is the model we consider next, and throughout the rest of this lecture. Monotone circuits are defined in the same way as usual circuits except we do not allow the usage of NOT gates.

It seems intuitive that monotone circuits cannot calculate any function, because there is no way to simulate a NOT gate using AND and OR gates. We will formulate and prove a characterization of the functions that can be computed using monotone circuits:

Definition 2 (Monotone Function): $f : \{0,1\}^n \rightarrow \{0,1\}$ *is a* monotone function *if for every* $x, y \in \{0,1\}^n$, $x \geq y$ *implies* $f(x) \geq f(y)$. *Where the partial order* $\geq$ *on* $\{0,1\}^n$ *is the hamming order, i.e.,* $(x_1, \ldots, x_n) \geq (y_1, \ldots, y_n)$ *if and only if for every* $1 \leq i \leq n$ *we have* $x_i \geq y_i$.

Remark: The hamming partial order can be thought of as the containment order between sets, where a vector $x \in \{0,1\}^n$ corresponds to the set $S_x = \{i \mid x_i = 1\}$. Then: $x \leq y$ if and only if $S_x \subseteq S_y$.

An example of a monotone function is $CLIQUE_{n,k} : \{0,1\}^{\binom{n}{2}} \rightarrow \{0,1\}$. The domain of the function $CLIQUE_{n,k}$ is the set of graphs on n vertices $\{1, \ldots n\}$. A graph is represented by assignments to the $\binom{n}{2}$ variables $x_{i,j}$, where for every pair $i, j \in \{1, \ldots n\}$, $x_{i,j} = 1$ iff (i,j) is an edge in the graph.

$CLIQUE_{n,k}$ is 1 on a graph if and only if the graph has a clique of size k. Clearly, $CLIQUE_{n,k}$ is a monotone function, because when our ordering is interpreted as the containment ordering between the sets of edges in a graph, then if a graph G contains a clique of size k, any other graph containing the edges of G will also contain the same clique.

Theorem 3 *A function* $f : \{0,1\}^n \rightarrow \{0,1\}$ *is monotone if and only if it can be computed by a monotone circuit.*

Proof:

- ($\Longrightarrow$) We will build a monotone circuit that computes f: For every α s.t. $f(\alpha) = 1$ we define:
$$\phi_\alpha(x) = \bigwedge_{\alpha_i = 1} x_i$$
We also define:
$$\phi(x) = \bigvee_{f(\alpha)=1} \phi_\alpha(x)$$
It is clear that ϕ can be realized as a monotone circuit. Now we claim that $\phi = f$.
  1. For every α s.t. $f(\alpha) = 1$, we have $\phi_\alpha(\alpha) = 1$ and therefore $\phi(\alpha) = 1$.
  2. If $\phi(x) = 1$, then there is an α s.t., $\phi_\alpha(x) = 1$ and thereby $f(\alpha) = 1$. The fact that $\phi_\alpha(x) = 1$ means that $x \geq \alpha$ by the definition of ϕ_α. Now, from the monotonicity of f we conclude that $f(x) \geq f(\alpha) = 1$, meaning $f(x) = 1$.
- ($\Longleftarrow$) The functions AND and OR and the projection function $p_i(x_1, \ldots, x_n) = x_i$ are all monotone. We will now show that composition of monotone functions forms a monotone function, and therefore conclude that every monotone circuit computes a monotone function.

  Let $g : \{0,1\}^n \to \{0,1\}$ be a monotone function. Let $f_1, \ldots, f_n : \{0,1\}^N \to \{0,1\}$ be also monotone. We claim that $G : \{0,1\}^N \to \{0,1\}$ define by:
$$G(x) = g(f_1(x), \ldots, f_n(x))$$
  is also monotone.

  If $x \geq y$ then from the monotonicity of $f_1, \ldots, f_n$, we have that for all i: $f_i(x) \geq f_i(y)$. In other words:
$$(f_1(x), \ldots f_n(x)) \geq (f_1(y), \ldots f_n(y))$$
  Now, from the monotonicity of g, we have:
$$G(x) = g(f_1(x), \ldots, f_n(x)) \geq g(f_1(y), \ldots, f_n(y)) = G(y)$$

■

We make analogous definitions for complexity in monotone circuits:

Definition 4 (*Mon-Size*, *Mon-Depth*): *Given a monotone function $f : \{0,1\}^n \to \{0,1\}$, we define:*

1. *Mon-Size*$(f) \stackrel{\text{def}}{=}$ *The minimum size of a monotone circuit computing f.*
2. *Mon-Depth*$(f) \stackrel{\text{def}}{=}$ *The minimum depth of a monotone circuit computing f.*

Obviously for every monotone function f, *Mon-Size*$(f) \geq$ *Size*(f), and *Mon-Depth*$(f) \geq$ *Depth*(f). In fact there are functions for which these inequalities are strict. We will not prove this result here.

Unlike the general circuit model, several lower bounds were proved for the monotone case. For example, it is known that for large enough n and specific k (depending on n):
$$\begin{aligned} \textit{Mon-Size}(CLIQUE_{n,k}) &= \Omega(2^{\frac{1}{3}n}) \\ \textit{Mon-Depth}(CLIQUE_{n,k}) &= \Omega(n) \end{aligned}$$

¿From now on we shall concentrate on proving a lower bound on *st-Connectivity*:

Definition 5 (*st-Connectivity*): *Given a directed graph G on n nodes, two of which are marked as s and t, st-Connectivity$(G) = 1$ if and only if there is a directed path from s to t in G.*

Obviously *st-Connectivity* is a monotone function since if we add edges we cannot disconnect an existing path from s to t.

Theorem 6 *Mon-Depth(st-Connectivity)* $= \Theta(\log^2(n))$

In a previous lecture, we proved that *st-Connectivity* is in NC_2. This we proved by constructing a circuit that performs $O(\log(n))$ successive boolean matrix multiplications. Notice that the operation of multiplying boolean matrices is a monotone operation (it uses only AND and OR gates). Therefore, the circuit constructed for *st-Connectivity* is actually monotone. If we define $Mon\text{-}NC_i$ to be the natural monotone analog of NC_i, then *st-Connectivity* is in $Mon\text{-}NC_2$. Also, from the above theorem *st-Connectivity* is not in $Mon\text{-}NC_1$. This gives us:

Corollary 7 $Mon\text{-}NC_1 \neq Mon\text{-}NC_2$

An analogous result in the non-monotone case is believed to be true, yet no proof is known.

We will proceed by reducing the question of monotone depth to a question in communication complexity.

3 Communication Complexity and Circuit Depth

There is an interesting connection between circuit depth and communication complexity which will assist us when proving our main theorem. Since the connection itself is interesting, we will prove it for general circuits. First some definitions:

Definition 8 *Given $f : \{0,1\}^n \to \{0,1\}$ we define a communication game G_f:*

- *Player 1 gets $x \in \{0,1\}^n$, s.t. $f(x) = 1$.*
- *Player 2 gets $y \in \{0,1\}^n$, s.t. $f(y) = 0$.*

Their goal is to find a coordinate i s.t. $x_i \neq y_i$.

Notice that this game is not exactly a communication game in the sense we defined in the previous lecture, since the two players do not compute a function, but rather a relation.

We denote the communication complexity of a game G by $CC(G)$. The connection between our complexity measures is:

Lemma 3.1 $CC(G_f) = Depth(f)$

Proof:

1. First we'll show $CC(G_f) \leq Depth(f)$. Given a circuit C that calculates f, we will describe a protocol for the game G_f. The proof will proceed by induction on the depth of the circuit C.
   - **base case:** $Depth(f) = 0$. In this case, f is simply the function x_i or $\neg x_i$, for some i. Therefore there is no need for communication, since i is a coordinate in which x and y always differ.

- **Induction step:** We look at the top gate of C: Assume $C = C_1 \wedge C_2$, then

$$Depth(C) = 1 + \max\{Depth(C_1), Depth(C_2)\}$$
$$\Downarrow$$
$$Depth(C_1), Depth(C_2) \leq Depth(C) - 1$$

  Denote by f_1 and f_2 the functions that C_1 and C_2 calculate respectively. By the induction hypothesis:

$$CC(G_{f_1}), CC(G_{f_2}) \leq Depth(C) - 1$$

  We know that $f(x) = 1$ and $f(y) = 0$, therefore:

$$f_1(x) = f_2(x) = 1$$
$$f_1(y) = 0 \text{ or } f_2(y) = 0$$

  Now, as the first step in the protocol, player 2 sends a bit specifying which of the functions f_1 or f_2 is zero on y. Assume player 2 sent 1. In this case they both know:

$$f_1(y) = 0$$
$$f_1(x) = 1$$

  And now the game has turned into the game G_{f_1}. This we can solve (using our induction hypothesis) with communication complexity $CC(G_{f_1}) \leq Depth(f_1)$. If player 2 sent 2 we would use the protocol for G_{f_2}. We needed just one more bit of communication. Therefore our protocol will have communication complexity of:

$$\begin{aligned} CC(G_f) &\leq 1 + \max\{CC(G_{f_1}), CC(G_{f_2})\} \\ &\leq 1 + \max\{Depth(f_1), Depth(f_2)\} \\ &= 1 + (Depth(f) - 1) \qquad = \quad Depth(f) \end{aligned}$$

  We proved this for the case where $C = C_1 \wedge C_2$. The case where $C = C_1 \vee C_2$ is proved in the same way, expect player 1 is the one to send the first bit (indicating if $f_1(x) = 1$ or $f_2(x) = 1$).

2. Now we'll show the other direction: $CC(G_f) \geq Depth(f)$. For this we'll define a more general sort of communication game based on two non-intersecting sets: $A, B \subseteq \{0,1\}^n$:

   - Player 1 gets $x \in A$
   - Player 2 gets $y \in B$
   - Their goal is to find a coordinate i s.t. $x_i \neq y_i$.

   We'll denote this game by $G_{A,B}$. Using the new definition G_f equals $G_{f^{-1}(1), f^{-1}(0)}$. We will prove the following claim:

   Claim 3.2 *If $CC(G_{A,B}) = d$ then there is a function $f : \{0,1\}^n \to \{0,1\}$ that satisfies:*

   - *$f(A) = 1$ (i.e., $f(x) = 1$ for every $x \in A$).*
   - *$f(B) = 0$*
   - *$Depth(f) \leq d$*

In the case of G_f, the function we get by the claim must be f itself, and we get that it satisfies $Depth(f) \leq CC(G_f)$, proving our lemma.

Proof: *(claim)* By induction on $d = CC(G_{A,B})$

- **Base case:** $d = 0$, meaning there is no communication, so there is a coordinate i in which all of A is different then all of B, and so the function $f(\alpha) = \alpha_i$ or the function $f(\alpha) = \neg\alpha_i$ will satisfy the requirements depending on whether the coordinate i is 1 or 0 in A.
- **Induction step:** We have a protocol for the game $G_{A,B}$ of communication complexity d. First assume player 1 sends the first bit in the protocol. This bit partitions the set A into two disjoint sets $A = A_0 \cup A_1$, or in other words, this bit turns our game into one of the following games (depending on the bit sent): $G_{A_0,B}$ or $G_{A_1,B}$. Each one of these has communication complexity of at most $d-1$ simply by continuing the protocol of $G_{A,B}$ after the first bit is already sent. Now, by the induction hypothesis we have two functions f_0 and f_1 that satisfy:
  - $f_0(A_0) = 1$ and $f_1(A_1) = 1$.
  - $f_0(B) = f_1(B) = 0$
  - $Depth(f_0), Depth(f_1) \leq d - 1$

  We define $f = f_0 \vee f_1$. Then:
  - $f(A) = f_0(A) \vee f_1(A) = 1$, because f_0 is 1 on A_0 and f_1 is 1 on A_1.
  - $f(B) = f_0(B) \vee f_1(B) = 0$
  - $Depth(f) = 1 + \max\{Depth(f_0), Depth(f_1)\} \leq d$

  So f is exactly what we wanted.

  If player 2 sends the first bit, he partitions B into two disjoint sets $B = B_0 \cup B_1$, and turns the game into G_{A,B_0} or G_{A,B_1}. By the induction hypothesis we have two functions corresponding to the two games, g_0 and g_1, so that:

  $$g_0(A) = g_1(A) = 1$$
  $$g_0(B_0) = g_1(B_1) = 0$$

  We define $g \stackrel{\text{def}}{=} g_0 \wedge g_1$. This g satisfies:
  - $g(A) = g_0(A) \wedge g_1(A) = 1$.
  - $g(B) = g_0(B) \wedge g_1(B) = 0$ (because g_0 is 0 on B_0, and g_1 is 0 on B_1).

■

■

4 The Monotone Case

Let us remember that our goal was to prove tight bounds on the monotone depth of *st-Connectivity*. Therefore we will define an analogue game for monotone functions, that will give us a lemma of the same flavor as the last.

4.1 The Analogous Game and Connection

Definition 9 (Monotone game): *Given a monotone* $f : \{0,1\}^n \to \{0,1\}$ *we define a communication game* M_f*:*

- *Player 1 gets* $x \in \{0,1\}^n$*, s.t.* $f(x) = 1$.
- *Player 2 gets* $y \in \{0,1\}^n$*, s.t.* $f(y) = 0$.

Their goal is to find a coordinate i *s.t.* $x_i > y_i$*, i.e.* $x_i = 1$ *and* $y_i = 0$ *We denote this kind of a game a* monotone game.

The game is exactly the same as G_f, except f is monotone, and the goal is more specific; i.e., the goal is to a find a coordinate i where not only $x_i \neq y_i$ but also $x_i > y_i$. Notice that the goal is always achievable, because if there is no such i, then y is at least as large as x in every coordinate. This means that $y \geq x$, but this contradicts the fact that f is monotone and $f(x) = 1$, $f(y) = 0$.

Our corresponding lemma for the monotone case is:

Lemma 4.1

$$CC(M_f) = Mon\text{-}Depth(f)$$

Proof: The proof is similar to the non-monotone case:

1. When building the protocol from a given circuit:
   - **Base case:** since f is monotone, if the depth is 0, we have that $f(\alpha) = \alpha_i$ and therefore it must be the case that $x_i = 1$ and $y_i = 0$. Hence, again there is no need for communication, and the answer is i (after all $x_i > y_i$).
   - **Induction step:** In the induction step, the top gate separates our circuit into two sub-circuits. The protocol then uses one communication bit to decide which of the two games corresponding to the two sub-circuits to solve. Since the sub-circuits are monotone, by the induction hypothesis they each have a protocol to solve their matching monotone game. This solves the monotone game corresponding to the whole circuit, since the sub-games are monotone, and therefore the coordinate i found satisfies $x_i > y_i$.
2. When building the circuit from a given protocol:
   - **Base case:** if there is no communication, both players already know a coordinate i in which $x_i > y_i$, hence our circuit would simply be $f(\alpha) = \alpha_i$, which is monotone and of depth 0.
   - **Induction step:** Each communication bit splits our game into two sub-games of smaller communication. Notice that if the original game was a monotone game, so are the two sub-games. By the induction hypothesis, the circuits for these games are monotone. Now, since we only add AND and OR gates, the circuit built is monotone.

$\blacksquare$

4.2 An Equivalent Restricted Game

Let us define a more restricted game than the one in Definition 9, that will be easier to work with. First some definitions regarding monotone functions:

Definition 10 (minterm, maxterm): *Let $f : \{0,1\}^n \to \{0,1\}$ be a monotone function.*

- *A* minterm *of f is $x \in \{0,1\}^n$ s.t. $f(x) = 1$ and for every $x' < x$ we have $f(x') = 0$.*
- *A* maxterm *of f is $y \in \{0,1\}^n$ s.t. $f(y) = 0$ and for every $y' > y$ we have $f(y') = 1$.*

For example, for *st-Connectivity*:

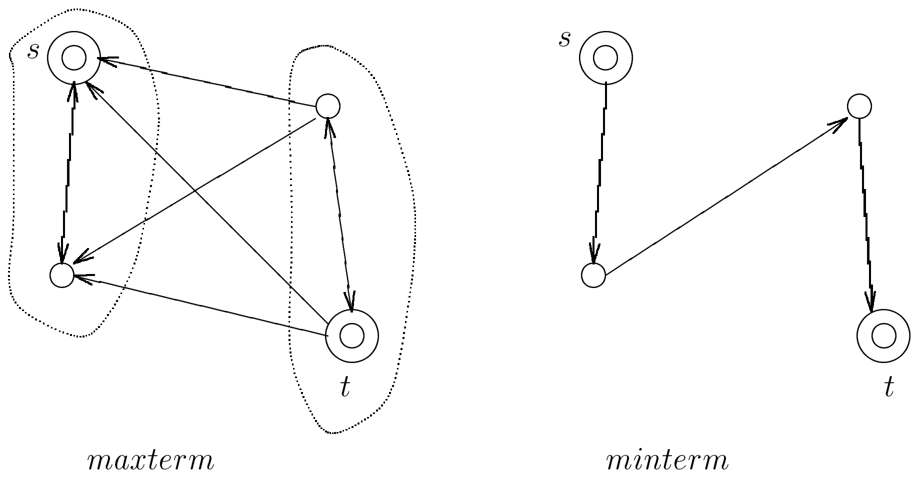

Figure 1: A maxterm and minterm example for *st-Connectivity*.

- The set of minterms is the set of graphs that contain only a simple (does not intersect itself) directed path from s to t:
  1. If a graph G is a minterm, then it must contain a simple path from s to t, and it cannot contain any other edge. This is because *st-Connectivity*$(G) = 1$, therefore there is a simple path P from s to t in G. G cannot contain any other edges, because then $P < G$ (in the edge containment order), but *st-Connectivity*$(P) = 1$, contradicting the fact that G is a minterm.
  2. Every G that is a simple path from s to t is a minterm, because *st-Connectivity*$(G) = 1$ and every edge we drop will disconnect s from t, therefore it is minimal.
- The set of maxterms for *st-Connectivity* is the set of graphs G s.t. G's set of vertices can be partitioned into two disjoint parts S and T that satisfy:
  1. $s \in S$ and $t \in T$.
  2. G contains all possible directed edges except those from S to T

  This is indeed the set of maxterms for *st-Connectivity*:

  1. If G is a maxterm then let S be the set of vertices that are reachable from s in G. Set T to be all other vertices. $t \in T$ because one cannot reach t from s in G, since

st-Connectivity$(G) = 0$. Also, G must contain all edges except those from S to T, otherwise we can add the missing edges and still leave t unconnected from s. There are no edges from S to T by the definition of S as the connected component reachable from s.

2. If G satisfies both criteria, then every path starting from s in G will remain in S and therefore will not reach t so *st-Connectivity*$(G) = 0$. Every edge we add to G will connect S to T and since S and T are strongly connected it will create a path between s and t.

Another way to view a maxterm of *st-Connectivity*, is that the partition is defined by a coloring of the vertices by two colors 0 and 1, where s is colored 0 and t is colored 1. The set of vertices colored 0 is S, and those colored 1 are T.

We will now use maxterms and minterms to define a new communication game:

Definition 11 ($\hat{M}_f$): *Given a monotone $f : \{0,1\}^n \to \{0,1\}$ we define a communication game $\hat{M}_f$:*

- *Player 1 gets $x \in \{0,1\}^n$, s.t. x is a minterm of f (in particular $f(x) = 1$).*
- *Player 2 gets $y \in \{0,1\}^n$, s.t. y is a maxterm of f (in particular $f(y) = 0$).*

Their goal is to find a coordinate i s.t. $x_i > y_i$, i.e. $x_i = 1$ and $y_i = 0$.

Notice that $\hat{M}_f$ is a restriction of M_f to a smaller set of inputs, therefore the protocol that will solve M_f will also solve $\hat{M}_f$. Hence $CC(\hat{M}_f) \le CC(M_f)$. In fact, the communication complexity of the two games is exactly the same:

Proposition 4.2 $CC(\hat{M}_f) = CC(M_f)$

Proof: What is left to prove is that: $CC(\hat{M}_f) \ge CC(M_f)$. Given a protocol for $\hat{M}_f$ we construct a protocol for M_f of the same communication complexity.

1. Player 1 has x s.t. $f(x) = 1$. He now finds a minimal x' s.t. $x' \le x$ but $f(x') = 1$. This is done by successively changing coordinates in x from 1 to 0, while checking that $f(x')$ still equals 1. This way, eventually, he will get x' that is a minterm.
2. In the same manner player 2 finds a maxterm $y' \ge y$.

The players now proceed according to the protocol for $\hat{M}_f$ on inputs x' and y'. Since x' is a minterm, and y' is a maxterm, the protocol will give a coordinate i in which:

$$\begin{array}{lcll} {x'}_i = 1 & \Longrightarrow & x_i = 1 & \text{because } x' \le x \\ {y'}_i = 0 & \Longrightarrow & y_i = 0 & \text{because } y' \ge y \end{array}$$

The communication complexity is exactly the same, since we used the same protocol except for a preprocessing stage that does not cost us in communication bits. ■

Combining our last results we get:

Corollary 12 *Given a monotone function $f : \{0,1\}^n \to \{0,1\}$:*

$$Mon\text{-}Depth(f) = CC(\hat{M}_f)$$

5 Two More Games

As we have seen, when examining bounds on the monotone depth of *st-Connectivity*, we need only examine the communication complexity of the following game denoted KW (for Karchmer and Wigderson) which is simply a different formulation of $\hat{M}_{st\text{-}Connectivity}$:

Given n nodes and two special nodes s and t,

- Player 1 gets a directed path from s to t.
- Player 2 gets a coloring C of the nodes by 0 and 1, s.t. $C(s) = 0$ and $C(t) = 1$.
- The goal is to find an edge (v, w) on player 1's path s.t. $C(v) = 0$ and $C(w) = 1$.

First we will use this formulation to show an $O(\log^2(n))$ upper bound on $Mon\text{-}Depth(st\text{-}Connectivity)$ using a protocol for KW with communication complexity $O(\log^2(n))$:

Proposition 5.1 $CC(KW) = O(\log^2(n))$

Proof: The protocol will simulate binary search on the input path of player 1. In each step, we reduce the length of the path by a factor of 2, while keeping the invariant that the color of the first vertex in the path is 0, and the color of the last is 1. This is of course true in the beginning since $C(s) = 0$ and $C(t) = 1$.

The base case, is that the path has only one edge, and in this case we are done, since our invariant guarantees that this edge is colored as we want. Now, player 1 sends player 2 this edge. The communication cost is $O(\log(n))$.

If the path is longer, player 1 asks player 2 the color of the middle vertex in the path. This costs $\log(n) + 1$ bits of communication — the name of the middle vertex sent from player 1 to player 2 takes $\log(n)$ bits, and player 2's answer costs one more bit. If the color is 1, the first half of the path satisfies our invariant, since the first vertex is colored 0, and now the last will be colored 1. If the color is 0, we take the second half of the path. In any case, we cut the length of the path by 2 with communication cost $O(\log(n))$.

Since the length of the original path is at most n, we need $O(\log(n))$ steps until we reach a path of length 1. All in all, we have communication complexity of $O(\log^2(n))$. ■

We will now direct our efforts towards proving a lower bound of $\Omega(\log^2(n))$ for $Depth(st\text{-}Connectivity)$ via a lower bound for KW. For this we will continue to yet another reduction into a different communication game called $FORK$:

Definition 13 ($FORK$): *Given $n = l \cdot w$ vertices and three special vertices s, t_1, and t_2, where the n vertices are partitioned into l layers $L_1, \ldots, L_l$, and each layer contains w vertices:*

- *Player 1 gets a sequence of vertices $(x_0, x_1, \ldots, x_l, x_{l+1})$, where for all $1 \leq i \leq l$: $x_i \in L_i$, and $x_0 = s$, $x_{l+1} = t_1$.*
- *Player 2 gets a sequence of vertices $(y_0, y_1, \ldots, y_l, y_{l+1})$, where for all $1 \leq i \leq l$: $y_i \in L_i$, and $y_0 = s$, $y_{l+1} = t_2$.*
- *Their goal is to find an i such that $x_i = y_i$ and $x_{i+1} \neq y_{i+1}$.*

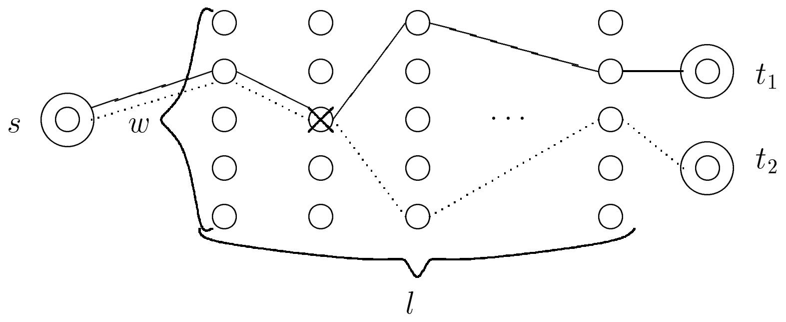

Figure 2: Player 1's sequence is solid, player 2's is dotted, and the fork point is marked with an x.

Obviously, such an i always exists, since the sequences start at the same vertex (s), and end in different vertices ($t_1 \neq t_2$), therefore there must be a fork point.

Note: The sequences the players get can be thought of as an element in $\{1, \ldots, w\}^l$. Since the start vertex is set to be s, and the end vertices for both players are also set to be t_1 and t_2 (depending on the player).

This game is somewhat easier to deal with then KW, because of the symmetry between the players. We will show that this new game needs no more communication than KW , and therefore, proving a lower bound on its communication complexity suffices.

Proposition 5.2 $CC(FORK) \leq CC(KW)$

Proof: Assuming we have a protocol for KW, we will show a protocol for $FORK$ which uses the same amount of communication. Actually, as in the proof of Proposition 4.2, all that the players have to do is some preprocessing, and the protocol itself does not change.

Recall that in the game KW player 1 has a directed path between two special vertices s and t, that goes through a set of regular vertices. Player 2 has a coloring of all vertices by 0 and 1, where s is colored 0, and t is colored 1.

To use the protocol for KW, the players need to turn their instance of $FORK$ into an instance of KW.

- We define the vertex s in $FORK$ to be s in KW.
- We define the vertex t_1 to be t.
- All other vertices are regular vertices.
- The path of player 1 remains exactly the same — it is indeed between s and $t(= t_1)$.
- The coloring of player 2 is: a vertex is colored 0 if and only if it is in his input path of vertices — note that s is colored 0, since it is the first vertex in his sequence, and $t(= t_1)$ is colored 1 because it is not on this path (which goes from s to t_2).

After this preprocessing we use the protocol for KW to get an edge (u, v) that is on player 1's path, where u is colored 0, and v is colored 1. This means, that u is on player 2's path, because it is colored 0, and v is not, because it is colored 1.

Hence, u is exactly the kind of fork point we were looking for, since it's in both players path and its successor is different in the two paths. ■

In the next lecture we will prove that

$$CC(FORK) = \Omega(\log(l)\log(w))$$

Setting $l = w = \sqrt{n}$, our main theorem follows:

$$CC(st\text{-}Connectivity) = \Theta(\log^2(n))$$