

## Lecture 4: NP-Completeness

Instructor: Dieter van Melkebeek

Scribe: Ozcan Ilikhan

Last time we introduced the model of a NTM, which we mentioned is not a realistic model of computation but nevertheless exactly captures the complexity of some important classes of problems. In particular, we started to turn our attention to the complexity class NP and briefly mentioned that this complexity class exactly characterizes those languages for which there exists an efficient<sup>1</sup> verification mechanism. In this lecture we continue our discussion of NP and its connection to efficient verifiability. We present the striking relationship between the complexity of almost all the problems known to be in NP but not known to be in P. To formalize our results we use the tools we developed in the previous class, namely reducibility and completeness.

## 1 Efficient Verification

**Definition 1.** We say that a language  $L$  has an efficient verifier if there exist  $c > 0$  and  $V \in P$  s.t.  $x \in L \iff (\exists y \in \Sigma^{\leq |x|^c}) \langle x, y \rangle \in V$ .

$V$  refers to a verification procedure that runs in polynomial time, and  $y$  refers to a short proof (i.e., a certificate, or a witness) for the membership of  $x$  in  $L$ .

It turns out that many problems of practical interest have this property of efficient verifiability. Before giving examples, we establish the connection between these class of problems and the class NP alluded to earlier.

**Theorem 1.**  $L \in \text{NP}$  iff  $L$  has an efficient verifier.

*Proof.*  $\Leftarrow$ ) Call  $L$ 's verifier  $V$ , and call  $V$ 's TM  $M_V$ . We describe a NTM  $N$  for  $L$  that runs in polytime. By definition, there is a constant  $c$  such that whenever  $x$  is in  $L$  there is a string  $y$  of length at most  $|x|^c$  such that  $\langle x, y \rangle$  is accepted by  $M_V$ .  $N$  simply “guesses” that string  $y$  and then (with polynomial overhead) simulates  $M_V$ .

$\Rightarrow$ ) Call  $L$ 's NTM  $N$ . By definition, a string  $x$  is in  $L$  iff  $N$  accepts  $x$  within  $|x|^c$  steps on some computation path,  $c$  being a constant. Then the description of any of those accepting paths is a certificate for  $x$ . Formally,  $V = \{\langle x, y \rangle \mid y \text{ is the sequence of nondeterministic choices that } N \text{ makes on an accepting branch of computation}\}$ . We see that  $V$  is in P: the TM for  $V$  simulates  $N$  on  $x$  by simulating its nondeterministic transitions in accordance with  $y$ , and accepts iff  $N$  accepts.  $\square$

Now we give some examples of problems that are efficiently verifiable (i.e., that are in NP):

- A boolean formula is in “conjunctive normal form” (CNF) if it is a conjunction of disjunctions of literals, where a literal is a boolean variable or its negation. Our first example of an efficiently verifiable language is  $\text{SAT} = \{\text{boolean formulas } \phi \text{ in CNF} \mid \exists \text{ an assignment that satisfies } \phi\}$ . Given a satisfiable  $\phi$ , the certificate for its membership in SAT is any satisfying assignment, which is clearly of polynomial (in fact linear) size in the length of  $\phi$ .

<sup>1</sup>Recall that we capture efficiency in a time-bounded setting by the class P.

This fundamental problem is a stripped down version of its more natural form, SEARCH-SAT, a commonly occurring problem in many fields such as artificial intelligence. SEARCH-SAT further asks, given  $\phi$ , a satisfying assignment if one exists. Even though at first sight SAT might appear to be too simple compared to SEARCH-SAT, a closer look reveals that they are equally difficult in the sense that, given  $\phi$ , if we know a solution for either problem then we can efficiently compute the solution for the other. So SAT exactly captures the inherent complexity of the SEARCH-SAT problem.

```

SEARCH-SAT()
(1)  if  $\phi \notin SAT$ 
(2)    return "NO"
(3)  foreach  $i = 1, \dots, n$ 
(4)    if  $\phi|_{x_i \leftarrow 0} \in SAT$ 
(5)       $x_i \leftarrow 0$ 
(6)    else
(7)       $x_i \leftarrow 1$ 
(8)  return "x"

```

We can argue this by using the notion of polynomial time oracle reductions: Clearly,  $SAT \leq_o^P SEARCH-SAT$ . We show that  $SEARCH-SAT \leq_o^P SAT$  by providing a procedure that, given  $\phi$ , uses a SAT-oracle to compute a solution for SEARCH-SAT as follows. Query the oracle with  $\phi$ , and if the answer is negative then  $\phi$  is not satisfiable. Else pick a variable in  $\phi$ , say  $x_i$ , and set it to any value, say 0. Now query the SAT-oracle to see if  $\phi$  with the value 0 substituted for  $x_i$  is satisfiable. If not, then set  $x_i$  to 1. Proceed with the remaining variables and gradually build up a satisfying assignment for  $\phi$ . The total number of oracle-queries is linear in the number of variables of  $\phi$ . So  $SAT \leq_o^P SEARCH-SAT$  and  $SEARCH-SAT \leq_o^P SAT$ , which is abbreviated as  $SAT \equiv_o^P SEARCH-SAT$ .

- It is often more convenient to work with a different version of satisfiability with formulas that have more structure. For this purpose, we define the language  $3SAT = \{\phi \mid \phi \text{ is a satisfiable boolean formula in 3-CNF form}\}$ . A formula is in 3-CNF form if it is in CNF form and each clause contains exactly three literals. We can also define SEARCH-3SAT, which will have the similar relation.  $3SAT \equiv_o^P SEARCH-3SAT$
- A "vertex cover" in a graph  $G$  is a subset of vertices such that every edge in  $G$  is incident to at least one vertex from that subset. The language  $VC = \{\langle G, k \rangle \mid \exists \text{ vertex cover for } G \text{ of size } \leq k\}$  is efficiently verifiable: the certificate for membership is the purported vertex cover.

Similar to the relationship between SAT and SEARCH-SAT, VC is the decision-version of its more naturally occurring form, namely the SEARCH-VC problem, which asks, given  $G$ , a vertex cover of minimal size. There are intermediate versions of this problem as well. One can develop the problem that just asks for a vertex cover of size  $\leq k$  for inputted  $k$ . One can develop the problem that asks for the minimum number of vertices needed to form a vertex cover. All versions are equivalent under polynomial reductions.

To show that  $VC \equiv_o^P SEARCH-VC$ , it is helpful to first consider an intermediate problem, namely the VC optimization problem, which asks, given  $G$ , the size of the minimal vertex

cover. Using a binary search, we query the VC-oracle to find the minimum number of vertices needed. Once we find the minimum size of a vertex cover, we can find it step by step using a similar method to the SAT problem. We pick a vertex and redraw the graph as if it were in the vertex cover. Then we query and ask if there is a vertex cover of size  $\leq k - 1$  on this new graph. If yes, we keep this vertex in the cover and continue. Otherwise we restore the graph to continue. We traverse the graph until we successfully remove the  $k$  vertices that result in the empty graph. This will be the vertex cover. So, SEARCH-VC is no harder than the optimization problem.

- SUBSET-SUM =  $\{\langle y_1, \dots, y_n, t \rangle \mid \exists I \subseteq \{1, \dots, n\} \text{ s.t. } \sum_{i \in I} y_i = t\}$  Given a bunch of integers and a target value, is there a subset of those integers that adds up to the target?

## 2 The P vs NP question

The three problems above are examples of literally thousands of problems that are encountered in everyday science and engineering, all of which have equivalent (in the sense of polytime Turing reductions) decision-versions that are in NP, and none of which have a known efficient solution. This is referred to as the P vs NP question: clearly  $P \subseteq NP$ , as any deterministic TM is also a NTM; but is  $P = NP$ ? This is one of the most important problems of contemporary mathematics due to its philosophical and practical significance. A philosophical interpretation of this question is whether being able to efficiently verify proofs implies the ability to efficiently come up with those proofs. When stated this way its answer seems obvious, nevertheless this question remains open. On the practical side, a positive answer to this question would have utopic consequences<sup>2</sup> for science and engineering, but would also bring the demise of public-key cryptography, as we will see later in the semester.

## 3 The NP vs co-NP question

Given a complexity class  $\mathcal{C}$ , we define  $\text{co-}\mathcal{C} = \{\bar{L} \mid L \in \mathcal{C}\}$ . The NP vs co-NP question relates to the question of whether nondeterministic computations can be easily complemented. Recall from last lecture the asymmetry between a nondeterministic computation that yields a “yes” answer and one that yields a “no” answer: on a given input, an NTM rejects only if every possible computation path rejects that input. Equivalently, a co-nondeterministic computation accepts an input iff every possible computation path accepts it.

The NP vs co-NP question asks whether having an efficient NTM for a language implies the existence of an efficient NTM for the complement of that language. Similar to the P vs NP question, the answer intuitively seems to be negative: it doesn’t look like we can take the question of membership in an NP-language and efficiently translate it into a question of membership in a co-NP language and vice versa. Nevertheless this problem is also open.

For example, with SAT, if there is a satisfying assignment to  $\phi$ , it is easy to verify. If there is no satisfying assignment, we don’t know of an efficient proof. The negation of an unsatisfiable statement is a tautology. So an equivalent question is whether there exists efficient proofs of tautologies.

---

<sup>2</sup>One might question the utility of being able to solve a problem in time, say  $O(n^{1000})$ , but historical evidence suggests that once a problem is pulled in to the class P, it is quickly trimmed down to friendlier exponents.

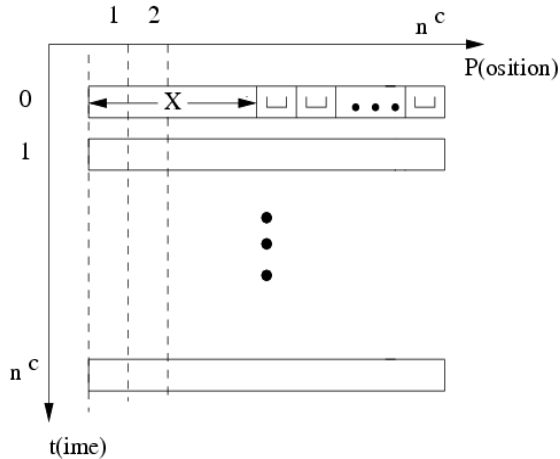


Figure 1: The computation tableau of  $M$  on input  $x$

The NP vs co-NP question relates to the topic of proof complexity, which can be summed up as the question of whether tautologies have short witnesses. Similar to the way that the satisfiability problem captures the complexity of the class NP as we discuss below, it follows that the problem of whether a formula is a tautology captures the complexity of the class co-NP. We will cover proof complexity at a later lecture.

Notice in passing that  $P = NP$  implies  $NP = \text{co-NP}$  but not necessarily the other way around.

## 4 NP-Completeness

It turns out nearly all the problems for which there is an efficient verification procedure but for which we don't know of an efficient solution, are different manifestations of the very same problem. That is, the inherent complexity behind them is so tightly connected that if one of them can be efficiently solved then so can all the rest.

The next theorem states the connection of the language SAT to the entire class NP under polynomial time mapping reductions. For didactic reasons, we show this result for a simplified model of computation, namely a single-tape TM with sequential access. The result extends to the random access model since we can simulate a random access machine on a sequential machine with a quadratic blowup in the running time of the former.

**Theorem 2.** *SAT is complete for NP under  $\leq_m^P$ . Further, each bit of the mapping reduction is computable in logarithmic space and polylogarithmic time.*

*Proof.* (Sketch) We already showed above that  $\text{SAT} \in \text{NP}$ . We now need to show that every problem in NP polynomial time mapping reduces to SAT. For this, fix a language  $L \in \text{NP}$ . Let  $M$  be an NTM that decides  $L$  and that runs in time  $n^c$  on an input of length  $n$ , where  $c$  is a constant. In the rest of the proof we discuss how, for any string  $x$ , the question of  $x$ 's membership in  $L$  can be efficiently translated into a question of satisfiability of a CNF formula,  $\phi_x$ . That is, given  $x \in \Sigma^n$  we construct a CNF formula  $\phi_x$  in polytime in  $n$  such that  $x \in L(M) \iff \phi_x \in \text{SAT}$ . We do this by considering the computation tableau of  $M$  on input  $x$ .

A *computation tableau* of  $M$  on input  $x$  depicts the contents of the tape of  $M$  in successive steps during a computation of  $M$  on  $x$ . Clearly, we need only consider the tape positions up to  $n^c$ , and time steps up to  $n^c$ .

$\phi_x$  comprises variables and clauses. We first describe its variables. For each time step, we have variables that describe the configuration of  $M$  at that time step. Therefore for all  $0 \leq t \leq n^c$ , and for all  $q \in Q$ , we have a boolean variable  $y_{t,q}^{(state)}$  which is true iff  $M$  is in state  $q$  at time step  $t$  while computing on  $x$ . In addition, for all  $0 \leq t \leq n^c$ , and for all  $1 \leq p \leq n^c$ , we have  $y_{t,p}^{(tapehead)}$  which is true iff  $N$ 's tape head is located at the  $p^{th}$  position of its tape at time step  $t$  while computing on  $x$ . Also, for all  $a \in \Sigma$  we have  $y_{t,p,a}^{(tapehead)}$ , which is true iff  $N$ 's tape contains the symbol  $a$  at the  $p$ th position of its tape at time step  $t$  while computing on  $x$ .

Next we describe the clauses of  $\phi_x$ . The clauses basically force the variables of  $\phi_x$  to have their intended meaning. That is, we set-up these clauses so that they are satisfiable iff there is a setting of the variables that describes a valid accepting computation of  $M$  on input  $x$ . To achieve this,  $\phi_x$  contains :

- i) clauses that capture the initial configuration of  $M$ ,
- ii) clauses that express the valid transitions of  $M$  (valid tape head movements and valid evolution of tape contents at each time step, in accordance with the transition relation  $\delta$  of  $M$ ), and
- iii) clauses that express the final accepting configuration of  $M$  at time step  $n^c$ .

There are many details involved in setting up these clauses, we briefly touch on some here and leave the rest to the reader.

For example, since the 0th row in the tableau corresponds to the initial configuration of  $M$  on  $x$ , we have the unit clauses  $(y_{0,q_0}^{(state)} \wedge \bigwedge_{i \neq 0} \bar{y}_{0,q_i}^{(state)})$  to demand that  $M$  be at its initial state before the first step of the computation. Also associated with the initial configuration we have the unit clauses  $(y_{0,0}^{(tapehead)} \wedge \bigwedge_{0 < p} \bar{y}_{0,p}^{(tapehead)})$  to specify the tape head position of  $M$ . In addition, we have  $(\bigwedge_{p > n} y_{0,p,\sqcup}^{(contents)} \wedge \bigwedge_{p > n, a \neq \sqcup, a \in \Sigma} \bar{y}_{0,p,a}^{(contents)})$  to require that  $M$ 's tape only contain blanks beyond the input  $x$ . Further, denoting the  $i$ th symbol of  $x$  as  $x_i$ , we have  $(\bigwedge_{p \leq n} y_{0,p,x_p}^{(contents)} \wedge \bigwedge_{p \leq n, a \neq x_p, a \in \Sigma} \bar{y}_{0,p,a}^{(contents)})$  to specify that the input is  $x$ .

Since the contents of a cell cannot change if the tape head was not on that cell the step before, for each  $1 \leq p \leq n$  and  $0 \leq t \leq n^c$  we have the clause  $(y_{t,p}^{(tapehead)} \vee \bigwedge_{a \in \Sigma} (y_{t,p,a}^{(contents)} \leftrightarrow y_{t+1,p,a}^{(contents)}))$  (we don't write it in CNF form here for readability.)

For the cell on which the tape head is positioned at time (i.e, row)  $t$ , there may be multiple possibilities for the next position of the tape head and the contents of the cell depending on the transition relation  $\delta$  of  $M$ . For example, if  $\{(q_2, b, R), (q_3, c, L)\} \subset \delta(q_1, a)$ , then for each  $1 \leq p \leq n$  and  $0 \leq t \leq n^c$  we have the clauses

$$\begin{aligned} & (y_{t,p}^{(tapehead)} \wedge y_{t,q_1}^{(state)} \wedge y_{t,p,a}^{(contents)}) \rightarrow \\ & (y_{t+1,p+1}^{(tapehead)} \wedge y_{t+1,q_2}^{(state)} \wedge y_{t+1,p,b}^{(contents)} \wedge \psi_R) \vee (y_{t+1,p-1}^{(tapehead)} \wedge y_{t+1,q_3}^{(state)} \wedge y_{t+1,p,c}^{(contents)} \wedge \psi_L), \end{aligned}$$

where  $\psi_R$  is  $(\bigwedge_{i \neq p+1, 1 \leq i \leq n^c} \bar{y}_{t+1,i}^{(tapehead)} \wedge \bigwedge_{j \neq q_2, j \in Q} \bar{y}_{t+1,j}^{(state)} \wedge \bigwedge_{s \neq b, s \in \Sigma} \bar{y}_{t+1,p,s}^{(contents)})$  and  $\psi_L$  is left to the reader. In the end, the formula  $\phi_x$  contains  $O(n^{2c})$  variables and is constructible in time

polynomial in the length of  $x$ . The freedom of setting the variables of  $\phi_x$  corresponds to  $M$ 's freedom of making nondeterministic choices while computing on  $x$ , and thus  $\phi_x$  is satisfiable iff there is a valid computation path of  $M$  that accepts  $x$ .

The resulting formula  $\phi_x$  has a very simple structure. Due to this structure the reader can verify that each individual bit of the formula can be computed very efficiently, namely in time polynomial in the length of the position(address) of the bit to be computed, or equivalently, in time polylogarithmic in the size of the input  $x$ .

This establishes the mapping reduction, and the proof is complete. □

## 4.1 Completeness for NQLIN

Next we present a theorem that shows a much tighter result on the complexity of reductions that connect certain problems in NP to the SAT problem. Namely, SAT is also complete for those problems in NP that can be solved in quasi-linear time, under quasi-linear time mapping reductions. Moreover, in proving the hardness part of the theorem, we do not make any simplifying assumptions on the model of computation like we did in the previous theorem.

We start by defining the complexity classes related to quasi-linear time in the obvious way.

- $QLIN = \cup_{c>0} DTIME(n(\log^c(n)))$
- $NQLIN = \cup_{c>0} NTIME(n(\log^c(n)))$

**Theorem 3.** *SAT is complete for NQLIN under  $\leq_m^{QLIN}$ . Further, each bit of the mapping reduction is computable in logarithmic space and polylogarithmic time.*

*Proof.* As in Theorem 2, we first show that  $SAT \in NQLIN$ , then discuss how any language in NQLIN can be reduced in quasi-linear time to SAT.

First, observe that SAT can be computed in quasi-linear time by a nondeterministic random access machine that first guesses in linear time a satisfying assignment and then evaluates the formula with the guessed assignment in quasi-linear time.

To show that SAT is NQLIN-hard, we begin by making a key observation. In principle, a quasi-linear-time nondeterministic machine  $M$  can access locations on non-index tapes that have addresses of quasi-linear length. We claim that without loss of generality, we can assume that these addresses are at most of logarithmic length. The reason is that we can construct a NTM  $M'$  that simulates  $M$  with only a constant factor overhead in time and satisfies the above restriction. For each non-index tape  $\tau$  of  $M$ ,  $M'$  uses an additional non-index tape  $\tau'$  on which  $M'$  stores a list of all (address,value) pairs of cells of  $\tau$  which  $M$  accesses and that have an address value of more than logarithmic length. During the simulation of  $M$ ,  $M'$  uses  $\tau$  in the same way as  $M$  does to store the contents of the cells of  $\tau$  with small addresses; it uses  $\tau'$  for the remaining cells of  $\tau$  accessed by  $M$ .  $M'$  can keep track of the (address,value) pairs on tape  $\tau'$  in an efficient way by using an appropriate data structure, e.g., sorted doubly linked lists of all pairs corresponding to addresses of a given length, for all address lengths used. Note that the list of (address,value) pairs is at most quasi-linear in size so the index values  $M'$  uses on  $\tau'$  are at most logarithmic.  $M'$  can retrieve a pair, insert one, and perform the necessary updates with a constant factor overhead in time by exploiting the power of nondeterminism to guess the right tape locations<sup>3</sup>. Thus,  $M'$  simulates  $M$

---

<sup>3</sup>A deterministic simulation would incur a logarithmic overhead in time, which would be fine for our purposes, but would require a more involved data structure like a balanced search tree.

with a constant factor overhead in time and only accesses cells on its tapes with addresses of at most logarithmic length.

Next, similar to the proof of Theorem 2, with each step in a computation of  $M'$  we associate a block of boolean variables. Each block represents the following information at the beginning of a particular time step:

- i) the internal state of the machine,
- ii) the configuration (i.e, the contents and the tape head position) of all index tapes,
- iii) the tape head positions of all non-index tapes,
- iv) the contents of each cell that is under a tape head, and
- v) the transition that the machine is about to take at that step. Notice that this information is all that is needed to advance  $M'$ 's execution from that particular step to the next.

Each block needs to contain only  $O(\log n)$  many variables to be able to capture its intended information: i), iv) and v) can be represented by a constant number of variables, while in light of our earlier observation ii) and iii) can be described by  $O(\log n)$  many variables.

We use these blocks of variables to set up clauses in such a way that the clauses are satisfied iff the blocks represent a valid accepting computation of  $M'$  on a given input. We achieve this by checking:

- i) that the initial block corresponds to a valid transition out of an initial configuration of  $M'$ ,
- ii) that all pairs of successive computation steps are consistent in terms of the internal state of  $M'$ , the contents of the index tapes, and the tape head positions of all tapes that are not indexed,
- iii) that the accesses to the indexed non-input tapes are consistent,
- iv) that the accesses to the input tape are consistent with the input  $x$ , and
- v) that the final step leads to acceptance.

As in the proof of Theorem 2, conditions (i), (v), and each of the linear number of constituent conditions of (ii) can be expressed by clauses of polylogarithmic size using the above variables and additional auxiliary variables. Each bit of those clauses can be computed in polylogarithmic time and logarithmic space. All that remains is to show that the same can be done for conditions (iii) and (iv).

We check the consistency of the accesses to the indexed non-input tapes for each tape separately. For tape  $\tau$ , one (inefficient) way to perform the consistency check is to look at all pairs of blocks and verifying that, if they accessed the same cell of  $\tau$ , and if no other block in between accessed that cell, then the contents of that cell in the second block is as dictated by the transition encoded in the first block. While this approach captures the essence of the formulation, it isn't adequate for us due to the quadratic overhead it introduces.

This construction can be made efficient by first sorting the blocks, for each non-index tape  $\tau$ , in a stable way<sup>4</sup> on the value of the tape head location in each block. Then we can perform the

---

<sup>4</sup>A stable sort is one that exchanges the order of two elements only when it has to.

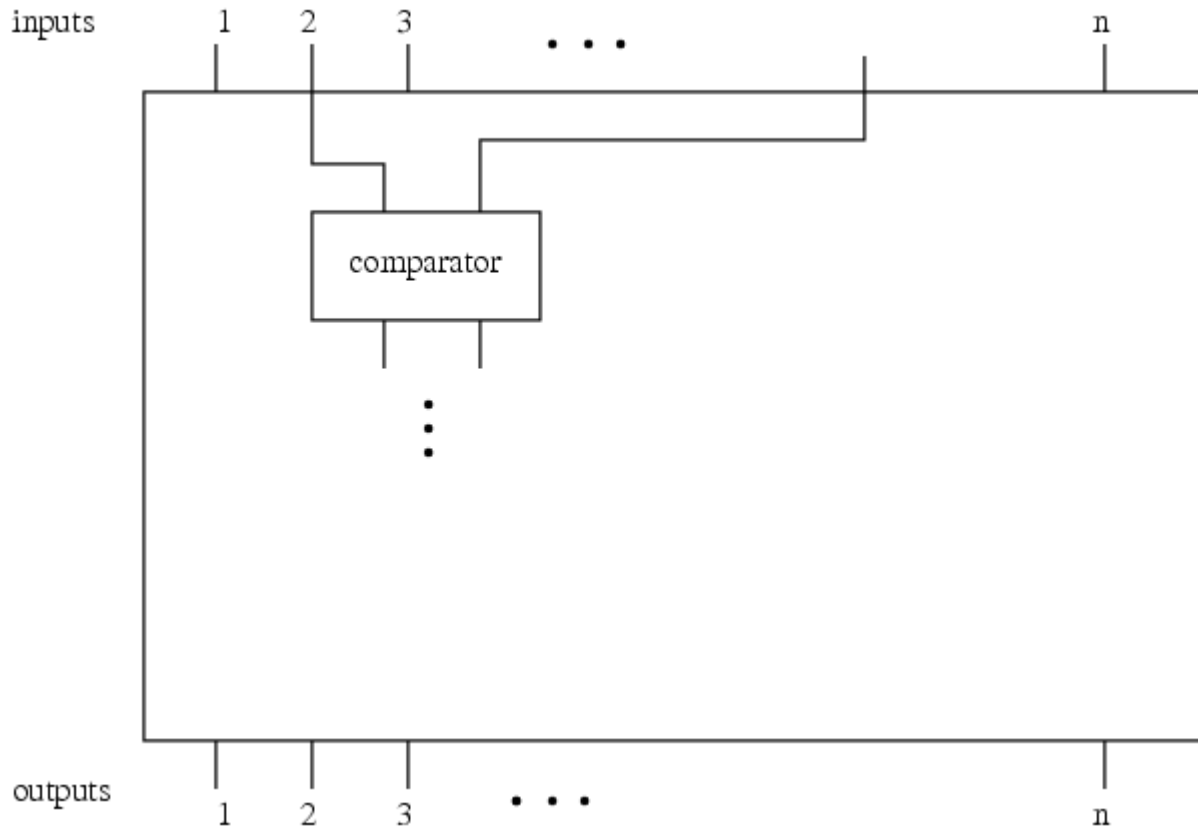


Figure 2: A simple diagram of a sorting network

consistency check for tape  $\tau$  by looking at all pairs of *consecutive* blocks and verifying that, if they accessed the same cell of  $\tau$ , the contents of that cell in the second block is as dictated by the transition encoded in the first block, and if they accessed different cells, then the contents of the cell in the second block is blank. These conditions can be expressed in the same way as (ii) above. Note that a stable sort is necessary in order to maintain the order of accesses to the same tape cell.

We now construct boolean clauses that mimic a deterministic sorting procedure. Since we are avoiding quadratic overheads, we focus on any  $n \log n$  sorting procedure. At this point, it may look as if we hit a block: in order to efficiently formulate a quasi-linear computation, we need to efficiently formulate another quasi-linear computation. We are not stuck, however; the latter computation is of a very specific type, one which lends itself to quasi-linear formulations, as we see next.

We formulate the sorting procedure by making use of *sorting networks*. Sorting networks are a specific type of circuit with  $n$  inputs and  $n$  outputs which, given  $n$  input values, each of length  $\log n$ , outputs those inputs in stable-sorted order. The circuit consists of a single type of element, called comparator element, which is basically a stable-sorting box for two elements. See Figure 2.

Without going into further detail we use the fact that there exist easily computable sorting networks of size  $O(n \log^2 n)$ . There are a number of constructions that yield this result, among which Batcher's networks are worth mentioning due to their simplicity. These are built using



the merge-sort divide-and-conquer strategy, where each (so-called odd-even) merger network is constructed using another level of divide-and-conquer. We refer the reader to the algorithms book CLRS for more on sorting networks.

We associate a block of boolean variables with each connection in the network and include clauses that enforce the correct operation of each of the comparator elements of the network. The latter conditions can be expressed in a similar way as condition (ii) above. The size and constructibility properties of the network guarantee that the resulting Boolean formula is of quasi-linear size and such that each bit can be computed in polylogarithmic time and logarithmic space.

The consistency of the input tape accesses with the actual input  $x$  can be checked in a similar way as condition (iii). The only difference is that before running the stable sorting for the input tape, we prepend  $n$  dummy blocks, the  $i$ th of which has the input tape head set to location  $i$ . The approach for handling condition (iii) then enforces that all input accesses are consistent with the values encoded in the dummy blocks. Since we know explicitly the variable that encodes the  $i$ th input bit in the dummy blocks, we can include simple clauses that force that variable to agree with the  $i$ th bit of  $x$ .

So we have established a quasi-linear mapping reduction from any problem in NQLIN to the SAT problem, and the proof is complete.

□

## 5 Next lecture

In the next lecture we will conclude NP-completeness and move on to Time-Bounded Nondeterminism to address questions that arose in this lecture.

## Acknowledgements

In writing the notes for this lecture, I perused the notes by Baris Aydinlioglu for lecture 3 from the Spring 2007 offering of CS 810, and the notes by Andrew Bolanowski for lecture 3 from the Spring 2010 offering of CS 710.