1 Nondeterminism as a Resource

The resources we've studied thus far are time and space. These resources correspond to performace measures for computers in the real world - runtime in the case of time, and memory usage in the case of space. In this section of the course, we broaden our notion of resource by considering *nondeterminism*, which is not known to be physically implementable. However it is a very useful notion in terms of achieving a deeper understanding of which problems are "feasible" and which are not.

Just as the multi-tape Turing machine captured deterministic computation, we define a notion of non-deterministic multi-tape Turing machine (NTM) to capture non-deterministic computation. Formally, a k-tape NTM is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_f)$, where all these symbols have the same interpretation as with a multi-tape TM, except that δ is now a *relation*, i.e., a subset of $Q \times \Gamma^k \rightarrow Q^k \times \{L, R, S\}$ rather than a partial function. The fact that δ is a partial function for deterministic multi-tape TMs means that every configuration of a multi-tape TM can have at most one next configuration; by allowing δ to be a relation, we allow any given configuration to have *several* possible next configurations. The choice of which configuration to go to next is interpreted as a "guess" or "nondeterministic choice" of the machine.

More formally, let C_i be the current configuration of the NTM. Assume the finite control is in some state $q \in Q$, and that the symbols being read on the k tapes are $a_1, a_2 \ldots a_k \in \Gamma$. If $(q, a_1, a_2, \ldots a_k, q', a'_1, a'_2 \ldots a'_k, b_1, b_2 \ldots b_k) \in \delta$, where $a'_1, a'_2 \ldots a'_k \in \Gamma$ and $b_1, b_2 \ldots b_k \in \{L, R, S\}$, then the following configuration C' is an allowed next configuration for C_i: C' is the same as C_i except that each symbol a_i is overwritten with a'_i, the i'th tape head has moved one step in the direction b_i, and the new state is q'.

A computation of a NTM M is a sequence of configurations $C_0, C_1 \ldots C_t$, where C_0 is the initial configuration, C_t is a final configuration (i.e., accepting or rejecting) and for each $1 \leqslant i \leqslant t$, C_i is an allowed next configuration for C_{i-1}. A computation is accepting if the final configuration is accepting and rejecting otherwise. An NTM M accepts an input x if there is an accepting computation of M on x.

Non-deterministic Turing machines decide exactly the recursive languages - this is further evidence for the Church-Turing thesis. However the situation changes if resource constraints are imposed on NTMs. This is where the "power of non-determinism" really comes into play.

We now define time-bounded and space-bounded NTMs. Given a time function T, an NTM M runs within time T if for each input x, *every* computation of M halts within $T(|x|)$ steps. Similarly, given a space function S, an NTM M uses space at most S if for each input x, *every* computation of M uses space at most $S(|x|)$.

Now we are ready to define non-deterministic time and space classes.

Definition 1 *Given a function $T : \mathbb{N} \rightarrow \mathbb{N}$, $\mathsf{NTIME}(T)$ is the class of languages L for which there is an NTM M which runs within time T and decides L.*

Definition 2 *Given a function $S : \mathbb{N} \to \mathbb{N}$,* $\mathsf{NSPACE}(S)$ *is the class of languages L for which there is an NTM M which uses space at most S and decides L.*

As an informal example of the power of non-deterministic computation, consider the problem of checking whether a partially completed Sudoku puzzle can be extended to a valid solution. In general, there is no simple deterministic solution for this, but non-deterministically, one could just *guess* numbers in all the blank squares and then verify at the end that the numbers satisfy the constraints of Sudoku. This ability to make guesses is what gives non-deterministic computation its power.

An example with much more practical relevance is the SAT problem of deciding whether a Boolean formula in conjunctive normal form is satisfiable or not. This problem is not known to be doable in polynomial time deterministically but is easily seen to be in non-deterministic polynomial time - simply guess Boolean values for each variable and then evaluate the formula with the guessed values to verify that they do indeed satisfy the formula. If the formula is satisfiable, there will be *some* sequence of guesses that works, and hence an accepting computation of the corresponding NTM, while if the formula is unsatisfiable, all computations will reject. Guessing a polynomial number of bits can be done in polynomial time non-deterministically, and so can evaluating a formula on a given assignment.

A good intuitive way of picturing the behaviour of an NTM on an input is via its *computation tree*, where each node corresponds to a configuration. The root corresponds to the initial configuration, and the children of a node v correspond to the allowable next configurations for the configuration corresponding to v. Note that for any fixed NTM M, there is an absolute constant C which bounds the number of allowable next configurations for any configuration. Hence the computation tree is of bounded degree. The leaves of the computation tree correspond to accepting and rejecting configurations. An NTM M accepts on an input x iff its computation tree on x has at least one accepting leaf. If an NTM runs within time T on input x, then the depth of the tree is at most T.

2 Relationships among Nondeterministic and Deterministic Classes

There are some obvious relationships between deterministic and non-deterministic classes. Since deterministic TMs are special cases of NTMs, the following holds:

Proposition 3 *Let $T : \mathbb{N} \to \mathbb{N}$ be any function. Then* $\mathsf{DTIME}(T) \subseteq \mathsf{NTIME}(T)$*, and* $\mathsf{DSPACE}(T) \subseteq \mathsf{NSPACE}(T)$*.*

Also, since no machine operating in time T can write on more than $O(T)$ worktape cells, the following holds:

Proposition 4 *Let $T : \mathbb{N} \to \mathbb{N}$ be any function. Then* $\mathsf{NTIME}(T) \subseteq \mathsf{NSPACE}(T)$*.*

We next show an efficient simulation of non-deterministic time by deterministic space.

Theorem 5 *Let $T : \mathbb{N} \to \mathbb{N}$ be a space-constructible function. Then* $\mathsf{NTIME}(T) \subseteq \mathsf{DSPACE}(T)$.

Proof. Let $L \in \mathsf{NTIME}(T)$, where T is a space-constructible function, and let M be a non-deterministic machine deciding L in time T. We give a deterministic space-bounded machine M' deciding L in space $O(T)$. M' first uses the space-constructibility of T to compute T in unary on a separate work tape, in space $O(T)$.

The basic idea of the simulation is for M' to cycle over all possible sequences of length T of non-deterministic choices of M and check whether any of these sequences leads to acceptance. The key is that cycling over all these sequences can be done in deterministic space T, since space can be re-used. Moreover, once all the non-deterministic choices are given, the remaining computation is a deterministic time T computation, which is trivially in space T.

More formally, let K be a constant which is an absolute upper bound on the number of allowable next configurations for any configuration of M. M' cycles over all K-ary sequences s of length T in lexicographic order (assuming an arbitrary order on the elements of $[K]$). Note that M' has already computed T, so it knows the length of the sequences over which it must cycle. For each sequence s, M' simulates M on x by using the elements of s as non-deterministic choices of M. It might so happen that a certain element of the sequence is not a valid non-deterministic move for M at that stage, in which case M' immediately rejects. Otherwise, it continues the simulation, storing on its worktapes the current configuration of M during the simulation, and updating the configuration according to the non-deterministic choices encoded by s. If the simulation of M according to the sequence s of non-deterministic choices ends in acceptance, M' accepts, otherwise it goes on to the next sequence. If no sequence results in acceptance, M' rejects.

Clearly M' accepts iff there is an accepting computation of M on x. The space usage of M' is $O(T)$, since at most space $O(T)$ is required to compute T, and then space $O(T)$ is used to store the current sequence of non-deterministic choices for M as well as the current configuration of M.

□

The next result is a strengthening of Theorem 4 in the previous lecture notes. It gives a simulation of non-deterministic space by deterministic time with exponential slowdown.

Theorem 6 *Let $S = \Omega(\log(n))$ be a space-constructible function. Then* $\mathsf{NSPACE}(S) \subseteq \bigcup_{c>1} \mathsf{DTIME}(c^S)$.

Proof. Let L be any language in $\mathsf{NSPACE}(S)$ and M be an NTM deciding L in space $O(S)$, where S is space-constructible. We define a deterministic TM M' deciding L in time c^S for some constant $c > 1$.

The basic idea is to formulate the question of whether M accepts an input x as a *graph reachability* question for a directed graph of size exponential in S. As there are efficient deterministic algorithms for graph reachability such as Depth-First Search (DFS) and Breadth-First Search (BFS) which operate in linear time, this implies that whether $x \in L$ can be decided in time exponential in S. In order to formulate the graph reachability question, we define the notion of *configuration graph* of an NTM M on an input x. The vertices of this graph are possible configurations of M on x - there are at most $(n+1)2^{O(S)}$ such vertices, by the same argument as in the proof of Theorem 4 in the previous lecture notes. There is an edge from a vertex u to a vertex v of the configuration graph if the configuration C_v corresponding to v is an allowable next configuration of C_u. Now the question of whether M accepts x is precisely the question of whether there is a path from the initial configuration C_0 to an accepting configuration in the configuration graph. Without loss of generality, we can assume that there is exactly one accepting configuration - the reason is that we can modify M so that if it accepts, it erases all the contents of its worktapes and moves the input head pointer to the first symbol of the input before accepting.

More formally, on input x, M' first uses the space-constructibility of S to compute S on a separate worktape in time at most exponential in S, using again Theorem 4 in the previous lecture notes. Once it knows S, it can construct explicitly on one of its worktapes a representation of the configuration graph of M on x. This again takes time at most exponential in S, since there are an exponential number of vertices and checking if a specific edge exists can be done in polynomial time. Then M' runs the BFS algorithm to check if the accepting configuration is reachable from the initial configuration in this graph. If yes, it accepts, otherwise it rejects. Clearly M' decides correctly whether M decides x and it does so in time exponential in S as long as $S = \Omega(\log(n))$. □

The most non-trivial result in this part of the lecture notes is Savitch's Theorem, which gives a quadratic simulation of non-deterministic space by deterministic space. Note that no analogue is known for time - to the best of our knowledge, non-deterministic time can be exponentially more powerful than deterministic time. Thus space as a resource has quite different properties than time. The key idea of the proof is to use a recursive divide-and-conquer approach and reuse space between different parts of the recursion.

Theorem 7 *[Savitch] Let $S = \Omega(\log(n))$ be a space-constructible function. Then $\mathsf{NSPACE}(S) \subseteq \mathsf{DSPACE}(S^2)$.*

Proof. Let $L \in \mathsf{NSPACE}(S)$ be a language, and let M be an NTM deciding L in space $O(S)$. We define a deterministic Turing machine M' deciding L in space $O(S^2)$.

Let x be the input to M. There is some constant $c > 1$ such that there at most $c^{S(|x|)}$ distinct configurations of M on x, using the argument in the proof of Theorem 4 in the previous lecture notes, together with the assumption that $S = \Omega(\log(n))$. Thus, if there is an accepting computation of M on x, there is one of length at most $c^{S(|x|)}$. M' will use a sub-routine $REACH(C, C', t)$ which

given as input configurations C and C' and a number t in binary, will return "true" iff there is a path from C to C' of length at most t in the configuration graph of M on x. To decide whether $x \in L$, M' simply calls $REACH(C_0, C_f, c^S)$ where C_0 is the initial configuration of M on x and C_f is the unique accepting configuration. We will show that REACH can be implemented to take space $O(S \log(t))$ in general, which is $O(S^2)$ when t is at most exponential in S.

Note that $REACH$ cannot store the configuration graph of M on x - that would take too much space, since the configuration graph has exponentially many nodes. Instead, $REACH$ will operate recursively in an *implicit* fashion, recomputing information as and when required. If $t = 1$, $REACH$ can easily check in space $O(S)$ if C' is an allowable next configuration for C. This is the base case of the recursion. When $t > 1$, the key idea is that if there is a path from C to C' of length at most t, there is a *middle* configuration D such that there is a path from C to D of length at most $\lceil t/2 \rceil$ and a path from D to C' of length at most $\lceil t/2 \rceil$. Of course $REACH$ does not know a priori what this middle configuration is. But it can cycle over all possible configurations D of M on x and for *each* one, check recursively that $REACH(C, D, \lceil t/2 \rceil)$ and $REACH(D, C', \lceil t/2 \rceil)$ both return "true". If this is the case, then $REACH$ returns "true", otherwise it tries another configuration for the middle configuration. If for each possible configuration D, either $REACH(C, D, \lceil t/2 \rceil)$ or $REACH(D, C', \lceil t/2 \rceil)$ returns "false", REACH returns "false".

The key to implementing $REACH$ space-efficiently is to note that space can be *re-used* between the two recursive calls of $REACH$. Thus the space usage of $REACH$ per level of recursion is simply $O(S)$ to store D as well as information about which recursive call it is executing. Since t goes down by a constant factor with each level of recursion, the number of levels of recursion is $O(\log(t))$, thus the total space usage of $REACH$ is $O(S \log(t))$.

$REACH$ has been described somewhat informally - by working out the details, one can construct a "Turing machine implementation" of $REACH$ which operates within the same space bound. □

We saw in the previous lecture notes that deterministic time and space complexity classes are closed under various natural operations such as union, intersection and complementation. How about non-deterministic classes? It's not hard to see that the same proofs as in the deterministic case show that non-deterministic time and space are closed under union and intersection as well.

Proposition 8 *For any time bound t, $\mathsf{NTIME}(t)$ is closed under union and intersection. The same closure results hold for space-bounded non-deterministic cases.*

We just sketch the proof for non-deterministic time being closed under union - the proofs for the other results are analogous. Let L_1 and L_2 be any two languages in $\mathsf{NTIME}(t)$. We need to show that $L_1 \cup L_2 \in \mathsf{NTIME}(t)$.Let M_1 and M_2 be NTMs deciding L_1 and L_2 respectively in time $O(t)$. We define an NTM M deciding L in time $O(t)$. M first runs M_1 on its input and then M_2, and

accepts if either accepts. It is easy to see that M has an accepting computation on its input if and only if either M_1 does or M_2 does.

Unlike in the deterministic case, however, closure under complementation is not known to hold for non-deterministic time. In the deterministic case, we just switched accepting and rejecting states to obtain closure under complementation. This doesn't work in the non-deterministic case - consider a computation tree for which half the leaves are accepting and half the leaves are rejecting, hence this tree corresponds to an input that is accepted by the machine. By switching accepting and rejecting states, we still get a tree with half the leaves accepting and half rejecting, i.e., the input will still be accepted by the machine.

Of course, it's possible that a different simulation could work to obtain closure under complementation. However, no such simulation is known, and in fact it is generally believed that non-deterministic time is not closed under complementation.

The situation is even more interesting for non-deterministic space. Here too the simple trick used in the deterministic case does not work to establish closure under complementation, and for a long time it was believed that non-deterministic space is not closed under complementation. However, in 1988, Immerman and Szelepcsenyi showed independently that this is true, using an argument which is elementary but rather subtle. This argument is beyond the scope of this course, but is available in a Supplementary Note on the web page in case you are interested.

Theorem 9 (Immerman, Szelepcsenyi) *Let $S = \Omega(n)$ be a space-constructible space bound. Then $L \in \mathsf{NSPACE}(S)$ iff $\bar{L} \in \mathsf{NSPACE}(S)$.*

How about hierarchies for non-deterministic time and space? For non-deterministic space, the same argument as for deterministic space gives a hierarchy. The key property of non-deterministic space used here is that it is closed under complementation - this is what allows us to "flip the diagonal" in the diagonalization argument. For non-deterministic time, however, this argument does not work as closure under complementation is unknown, and a different "indirect diagonalization" argument is required. The details are beyond the scope of this course.