CS 810: Complexity Theory	2/2/2007

Lecture 6: Alternation

Instructor: Dieter van Melkebeek — Scribe: Piramanayagam Arumuga Nainar

In the past few lectures, we studied in depth the non-deterministic model of computation. In this lecture, we will extend it to a more general model: alternation. We derive results for the classes of problems that can be solved using alternation. We also discuss three other characterizations of this new model. One of them is the alternating Turing machine, a generalization of non-deterministic Turing machines.

1 Motivation

Even though non-determinism is a hypothetical model, it is important because it captures the complexity of several interesting problems. For example, NP, the class of problems that can be solved in non-deterministic polynomial time contains languages whose membership can be verified efficiently (i.e. in polynomial time). Let us state that more formally:

$$\text{A language L} \in \text{NP iff there exists a } c > 0 \text{ and } V \in \text{P such that } x \in L \Leftrightarrow (\exists y \in \Sigma^{\leq |x|^c})\langle x, y\rangle \in V \tag{1}$$

However non-determinism is still too restrictive to efficiently solve some interesting problems. One such example is that of minimizing a CNF formula. Consider the equivalent decision problem of testing whether a CNF formula is of minimum size. Let us define MIN-CNF $= \{\phi | \phi$ is in CNF and is of minimum size$\}$. We could state membership in MIN-CNF as follows:

$$\phi \in \text{MIN-CNF} \Leftrightarrow (\forall y \text{ of size } < |\phi|)(\exists x)\phi(x) \neq y(x) \tag{2}$$

Any CNF y that is smaller than ϕ can certainly be expressed as a string of size polynomial in $|\phi|$. The same is the case for an assignment x of values to the variables in ϕ. Moreover, checking whether $\phi(x) = y(x)$ can be done in polynomial time (It takes two SAT verifications followed by a test of equality). So stmt. 2 can be rewritten (to better resemble the formulation of NP above) as follows:

$$\phi \in \text{MIN-CNF} \Leftrightarrow (\forall y \in \Sigma^{\leq |\phi|^c})(\exists x \in \Sigma^{\leq |\phi|^c})\langle \phi, y, x\rangle \in V \text{ for some } V \in \text{P} \tag{3}$$

If we look at the logical formulas used to describe membership in a language in NP(stmt. 1) and MIN-CNF(stmt. 3), we notice two significant differences: the latter has two quantifiers and one of them is an universal quantifier. This is the generalization we are going to make in the alternating model. If we wish to allow more than one quantifier in formulas expressing membership constraints in languages, they must be alternating between universal and existential quantifiers. If not, two adjacent quantifiers of the same type can be combined into one, leaving a quantified variable that is polynomial in the size of the input.

2 The Alternating Model

We define a new, general complexity class Q_k^p as the set of languages whose membership constraints can be expressed using formulas with k alternate existential and universal quantifiers, with each quantified variable of size polynomial in the length of the input, and the initial quantifier being existential for $Q = \Sigma$ and universal for $Q = \Pi$.

A more notational description of Σ_k^p and Π_k^p is given below:

Definition 1. $\Sigma_k^p = \{L |$ *membership in* L *can be expressed by a formula of the form* $x \in L \Leftrightarrow (\exists y_1 \in \Sigma^{\leq |x|^c})(\forall y_2 \in \Sigma^{\leq |x|^c}) \dots (Q y_k \in \Sigma^{\leq |x|^c})\langle x, y_1, y_2, \dots y_k\rangle \in V$ *for some* $V \in$ P *and some constant* $c\}$

Definition 2. $\Pi_k^p = \{L |$ *membership in* L *can be expressed by a formula of the form* $x \in L \Leftrightarrow (\forall y_1 \in \Sigma^{\leq |x|^c})(\exists y_2 \in \Sigma^{\leq |x|^c}) \dots (Q y_k \in \Sigma^{\leq |x|^c})\langle x, y_1, y_2, \dots y_k\rangle \in V$ *for some* $V \in$ P *and some constant* $c\}$

Notes: The quantifier Q for y_k in Σ_k^p is $\exists$ if k is odd and $\forall$ if k is even. The dual is true for Π_k^p. The super-script p in Σ_k^p and Π_k^p denotes poly-time verifiability. There is a small notational inconsistency. Σ is used to denote a class of problems as well as to denote the input alphabet. But the intended usage is usually clear from context.

2.1 Facts

If $k = 0$, there are no quantified variables and the verifier V can decide membership just by looking at the input. In other words, V is an algorithm for L and Σ_0^p is the same as P. In the absence of any quantifiers, there is no real distinction between existential and universal quantification. Thus, $\Sigma_0^p = \Pi_0^p =$ P.

The definition of Σ_1^p is exactly the same as that of NP given earlier and so, $\Sigma_1^p =$ NP. We will just state here, and prove a general result later, that $\Pi_1^p =$ coNP. And finally, the description of MIN-CNF matches that of Π_2^p, hence MIN-CNF $\in \Pi_2^p$

3 Class Hierarchy results

Proposition 1. $\Pi_{k-1}^p \subseteq \Pi_k^p$ *and* $\Sigma_{k-1}^p \subseteq \Sigma_k^p$

A simple result we can state for the classes introduced above is that we can solve more (or at least an equal number of) problems if we allow greater number of quantifier alternations. This is trivial because we can add a new, unused variable y_k and quantify it appropriately but leave V unchanged.

Proposition 2. $\Sigma_k^p \cup \Pi_k^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$

We can transform the membership constraint for Σ_k^p to that of: (1) Σ_{k+1}^p by adding an universally quantified variable that is not used by V in the end; (2) Π_{k+1}^p by adding an universally quantified variable that is not used by V in the beginning. Hence $\Sigma_k^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$. A similar argument applies for Π_k^p and hence the above result holds.

Proposition 3. $\Sigma_k^p = co\Pi_k^p$ *and* $\Pi_k^p = co\Sigma_k^p$

Proof. A language L is in $co\Pi_k^p$ iff we could verify in Π_k^p whether a string is not in L. This negates the membership constraint in Defn. 2. The quantifiers for $y_1, y_2, \dots y_k$ will get switched, leaving a Σ_k predicate. The verification task will be whether $\langle x, y_1, y_2, \dots y_k\rangle$ is not in V. Since $V \in$ P, we can do this efficiently. Thus, $co\Pi_k^p = \Sigma_k^p$. A similar argument holds for $\Pi_k^p = co\Sigma_k^p$ $\square$

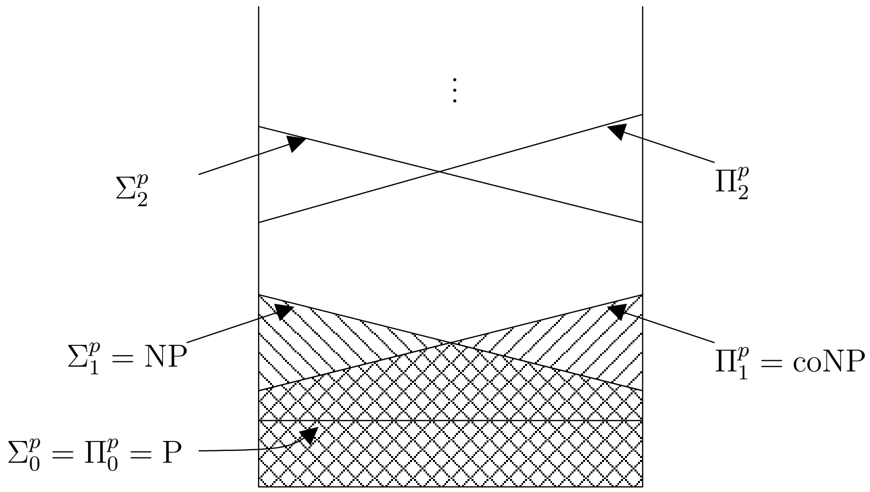

Figure 1: Pictorial illustration of the polynomial hierarchy

Propositions 1 and 2 can be illustrated using Figure 1. The k^{th} line sloping upwards bounds languages in Π_k^p and the k^{th} line sloping downwards bounds Σ_k^p. Both these lines are above the lines at the previous levels. For $k = 0$, the two lines collapse into each other, as $\mathrm{P} = co\mathrm{P}$.

Corollary 1. $\Pi_1^p = \mathrm{coNP}$, *since we already derived that* $\Sigma_1^p = \mathrm{NP}$

One question we can ask is whether the inclusion in Proposition 1 is strict. i.e. whether $\Sigma_k^p \subset \Sigma_{k+1}^p$. Even the simplest version of this question, for $k = 1$ (i.e. is $\mathrm{P} \subset \mathrm{NP}$ or in other words $\mathrm{P} \neq \mathrm{NP}$), has turned out to be hard, so far. The next result we prove is based on the definition of a polynomial hierarchy.

Definition 3. PH *is the class of all problems in the* polynomial time hierarchy *: any fixed number of alternations are allowed. i.e.* $\mathrm{PH} = \bigcup_k \Sigma_k^p$

Note: We need not include Π_k^p in the previous definition as a consequence of Proposition 2

Theorem 1. *If* $\Sigma_k^p = \Pi_k^p$ *for some* $k \geq 1$ *then* $\mathrm{PH} = \Sigma_k^p$, *in other words, the polynomial time hierarchy collapses to level* k.

Proof. Consider a language $L \in \Sigma_{k+1}^p$. A string $x \in L$ iff the following condition is true:

$$(\exists y_1 \in \Sigma^{\leq |x|^c})(\forall y_2 \in \Sigma^{\leq |x|^c}) \ldots (Q y_k \in \Sigma^{\leq |x|^c})(\overline{Q} y_{k+1} \in \Sigma^{\leq |x|^c})\langle x, y_1, y_2, \ldots y_k, y_{k+1}\rangle \in V \tag{4}$$

Now, $(\forall y_2 \in \Sigma^{\leq |x|^c}) \ldots (Q y_k \in \Sigma^{\leq |x|^c})(\overline{Q} y_{k+1} \in \Sigma^{\leq |x|^c})\langle x, y_1, y_2, \ldots y_k, y_{k+1}\rangle$ is a Π_k predicate on input $\langle x, y_1\rangle$. Because $\Sigma_k^p = \Pi_k^p$, there is an equivalent Σ_k predicate ϕ on input $\langle x, y_1\rangle$. The existential quantification in equation 4 can be merged with the initial existential quantifier of ϕ, thus leaving a new Σ_k predicate ϕ' on input $\langle x\rangle$. In other words, if $\Sigma_k^p = \Pi_k^p$, then every Σ_{k+1}^p language can be expressed using a Σ_k predicate. Thus, $\Sigma_{k+1}^p \subseteq \Sigma_k^p$. Combining this with proposition 1, we get $\Sigma_k^p = \Sigma_{k+1}^p$. Using multiple, alternate applications of the two operations: converting between Π_k and Σ_k predicates and merging like quantifiers together, we can prove that $\Sigma_j^p \subseteq \Sigma_k \forall j > k$. Thus, $\mathrm{PH} = \Sigma_k^p$. $\square$

Note: The condition $k \geq 1$ in the above theorem is required because we need at least one quantifier to make the premise $\Sigma_k^p = \Pi_k^p$ non-trivial.

Corollary 2. *If* $\mathrm{P} = \mathrm{NP}$ *then* $\mathrm{PH} = \mathrm{P}$.

Proof. If $\mathrm{P} = \mathrm{NP}$, we have already proved that $\mathrm{NP} = \mathrm{coNP}$. From the above theorem, the whole polynomial hierarchy collapses to NP and thus, to P. $\square$

¿From the above corollary, we can state that if PH does not collapse then $\mathrm{P} \neq \mathrm{NP}$. But the former is a stronger statement. That is, even if $\mathrm{P} \neq \mathrm{NP}$, the polynomial hierarchy may collapse to some other class. But the general conjecture in the community is that PH does not collapse. In fact there have been results that start from some assumption and derive that PH collapses in order to disprove the likeliness of that assumption.

4 Completeness

Now, we see an example of a problem that is complete in Σ_k^p.

Definition 4. *Let* $T.\Sigma_k$ *be the set of all true, fully quantified* Σ_k *formulas with a predicate that is a (1) CNF if* k *is odd and (2) DNF if* k *is even.*

Similarly, we can define $T.\Pi_k$ over the set of all true, fully quantified Π_k formulas.

Claim 1. $T.\Sigma_k$ *is* $\leq_m^p$*-complete for* Σ_k^p *and* $T.\Pi_k$ *is* $\leq_m^p$*-complete for* Π_k^p.

Proof. Consider a language $L \in \Sigma_k^p$ with associated verifier V. Suppose k is odd. Then the last quantifier is $\exists$, and the last quantifier together with V is $(\exists y_k \in \Sigma^{\leq |x|^c})V(\langle x, y_1, y_2, ..., y_k\rangle)$. This is an NP statement, so by the NP-completeness of SAT can be converted in polynomial time into the statement $(\exists z)\phi(\langle x, y_1, y_2, ..., y_{k-1}, z\rangle)$ for some CNF formula ϕ. Then $(\exists y_1 \in \Sigma^{\leq |x|^c})(\forall y_2 \in \Sigma^{\leq |x|^c})...(\exists z \in \Sigma^{\leq |x|^{c'}})\phi(\langle x, y_1, y_2, ..., y_{k-1}, z\rangle)$ is a a $T.\Sigma_k$ instance that is true if and only if $x \in L$.

Suppose k is even. Then the last quantifier is $\forall$, and the last quantifier together with V is $(\forall y_k \in \Sigma^{\leq |x|^c})V(\langle x, y_1, y_2, ..., y_k\rangle)$. This is a coNP statement, so by the coNP-completeness of TAUTOLOGIES can be converted in polynomial time into the statement $(\forall z)\phi(\langle x, y_1, y_2, ..., y_{k-1}, z\rangle)$ for some DNF formula ϕ. Given this, a $T.\Sigma_k$ instance is created as above.

As this reduction can be accomplished in polynomial time, $T.\Sigma_k$ is hard for Σ_k^p under $\leq_m^p$. As $T.\Sigma_k$ is in Σ_k^p by construction, $T.\Sigma_k$ is complete for Σ_k^p. A similar argument proves that $T.\Pi_k$ is Π_k^p-complete. $\square$

5 Alternate Characterizations

5.1 Using Oracle Machines

Claim 2. $\Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p}$. *In other words, the set of languages in the* $(k+1)^{th}$ *level of the polynomial hierarchy is the set of languages recognized by non-deterministic machines with access to oracles at the* k^{th} *level.*

Proof. For $k = 0$, the statement is $\text{NP} = \text{NP}^{\text{P}}$. $\text{NP} \subseteq \text{NP}^{\text{P}}$ is trivial because we can simply choose to not use the oracle. $\text{NP}^{\text{P}} \subseteq \text{NP}$ because the time complexity of the oracle will, in worst case, increase the degree of the polynomial in the time complexity.

The statement is non-trivial for $k > 0$. Consider $k = 1$. Suppose we have a language $L \in \Sigma_2^p$. Then, for some $c > 0$ and $V \in \text{P}$,

$$x \in L \Leftrightarrow (\exists y_1 \in \Sigma^{\leq |x|^c})(\forall y_2 \in \Sigma^{\leq |x|^c})\langle x, y_1, y_2 \rangle \in V$$

We can construct a non-deterministic TM M that guesses the value of y_1 and tries to solve $(\forall y_2 \in \Sigma^{\leq |x|^c})\langle x, y_1, y_2\rangle \in V$. The latter is a Π_1 formula and can be solved using an oracle for Σ_1^p because $\Sigma_1^p = co(\Pi_1^p)$ and in general any language can be solved given an oracle to its complement. Thus, $L \in \text{NP}^{\Sigma_1^p}$.

Suppose $L \in \text{NP}^{\Sigma_1^p}$. We can express that L is decidable in Σ_2^p as follows:

1. Express the computation path followed by the base NP machine as a Σ_1 formula by guessing the queries made by the machine, as well as the results of the queries.

2. Express the constraints that positive query responses are valid, again by using a Σ_1 predicate.

3. Express the constraints that negative query responses are valid by using a Π_1 predicate. The universal quantifier is required because in this step, we want to ensure the non-existence of a witness to the query, rather than its existence.

Overall a Σ_2 predicate expresses the decidability of L, and $L \in \Sigma_2^p$. Thus, $\Sigma_2^p = \text{NP}^{\Sigma_1^p}$. (The right hand side can also be written as NP^{NP}.)

Arguments for $k > 2$ are similar. □

5.2 Using Boolean Circuits

Claim 3. *Any language $L \in \Sigma_k^p$ can be expressed as an exponential size boolean circuit, with $k+1$ alternating levels of AND and OR gates of unbounded fan-in, with the last level having polynomial bounded fan-in and such that each bit of the circuit description can be computed in polynomial time, and vice-versa.*

Proof. Consider a language $L \in \Sigma_k^p$ with verifier V. We can construct an enormous boolean circuit as follows: an exponential number of polynomial circuits evaluating membership in V for all possible combinations of $y_1, y_2, \ldots y_k$ and a specific value for x. We can make these polynomial circuits to have only two levels by expressing the function $V(x, y_1, y_2, \ldots y_k)$ in CNF or DNF. The outputs of these circuits are combined hierarchically at k levels to leave a single output at the top-most level. This output is the decision whether $x \in L$. The i^{th} level contains an array of AND gates if y_i is universally quantified and an array of OR gates if y_i is existentially quantified. The number of gates in the i^{th} level is equal to the number of choices for inputs $y_1, y_2, \ldots y_{i-1}$. By appropriately choosing a CNF or DNF for the verifier V, we can merge the gates from the k^{th} level with the top gates from the normal form that is chosen. It can be verified that each bit is poly-time computable.

Now, consider a boolean circuit as described above with an OR gate at the top-most level. We must construct a Σ_k formula to compute the language decided by the circuit, i.e. we need to determine how many bits to guess for the $\exists$ and $\forall$ quantifiers and what the V will be. Consider the top-most OR gate. Its output will be 1 if there exists a gate at the lower level whose output is 1. We can express this as

$$(\exists G_2)(\text{output of gate } G_2 \text{ is } 1) \tag{5}$$

where G_2 is a second level gate. G_2 will be an AND gate. It will produce an output 1 if all the OR gates at the third level produce an output 1. So, stmt. 5 becomes:

$$(\exists G_2)(\forall G_3)(\text{output of gate } G_3 \text{ is } 1) \tag{6}$$

where G_3 is any gate whose output is connected as input to the chosen G_2. Since the circuit is of exponential circuit, we can represent gates in the circuit using polynomial size indices. If we repeatedly apply these steps, we will get a Σ_k predicate with the final verification task being that of evaluating the output of the gate at the $(k+1)^{th}$ level. Since each bit of the circuit description is computable in polynomial time, we can use the choices of the indices at each level to identify the gate that needs to be evaluated as well as the bits from the input that go into that gate. Thus, we can evaluate the last predicate of the Σ_k formula in poly-time. Thus, we have a Σ_k^p language corresponding to the circuit. Similarly we can construct a Π_k^p language corresponding to a circuit with an AND gate at the top-most level. $\square$

5.3 Alternating Turing Machines

In this section, we extend the non-deterministic Turing machine to model alternation. An alternating TM is a non-deterministic TM in which each non-halting state has an additional property: whether it is existential or universal. A configuration of the machine is accepting if either:

1. This is a halting configuration and the machine is in an accepting state, or
2. The current state is existential and at least one transition from this state leads to an accepting configuration, or
3. The current state is universal and all configurations from this state leads to an accepting configuration

The machine itself accepts an input if the initial configuration is accepting. The following gives our final characterization of Σ_k^p and Π_k^p.

Claim 4. $\Sigma_k^p = \{L | L$ *is accepted by an alternating TM with an existential start state that runs in polynomial time and has at most* $k-1$ *quantifier alternations*$\}$
$\Pi_k^p = \{L | L$ *is accepted by an alternating TM with an universal start state that runs in polynomial time and has at most* $k-1$ *quantifier alternations*$\}$

Proof. Consider a language $L \in \Sigma_k^p$. We can construct a k stage alternating TM M with the i^{th} stage guessing the value of y_i. To match the quantifier of the y_i's, all the states in the i^{th} stage must be existential if i is odd and universal if i is even. The verifier V of L does not require any non-determinism. So, the states that simulate V can be made existential or universal depending on k. Thus, M recognizes L and has an existential start state and performs at most $k-1$ alternations. Similarly, we can construct an alternating TM that recognizes a language in Π_k^p and satisfies the above constraints.

Consider an alternating TM M that performs at most $k-1$ alternations. We can construct an equivalent Σ_k^p or Π_k^p language L as follows. If M halts in polynomial time, the time it spends in each stage of the alternation is also polynomial. We can model the choices made by M in the i^{th} step as a polynomial length string y_i. If i existential (universal) stage, then y_i is quantified existentially (universally). The verifier V has to verify that the choices made are valid for the given input and machine M and also that the final state is halting and accepting. This can be done in polynomial time. The resulting formula is a Σ_k formula if M's initial state is existential, and is a Π_k formula otherwise. □

6 Time and Space

Using the same definition of time and space required by a Turing machine we can define complexity classes Σ_k-TIME(t) and Σ_k-SPACE(s). We can also derive hierarchy results using the same technique used earlier, namely delayed diagonalization. These results get simplified if we allow an unlimited number of alternations. We can define ATIME(t) as the set of problems that can be solved in time t on an alternating TM with any number of quantifier alternations. ASPACE(s) is the set of all problems that can be solved using space s on an alternating TM with any number of quantifier alternations.

Some results on Time and Space

Theorem 2. NSPACE(s) $\subseteq$ *ATIME*(s^2)

Proof. This follows from our proof of NSPACE(s) $\in$ DSPACE(s^2) in last lecture. Remember that we constructed a formula very similar to a Σ_k formula in our divide-and-conquer formulation of that proof. The existential quantifier guessed an intermediate configuration ($O(s)$ long) of the Turing machine and the universal quantifier was used to specify two independent reachability conditions (1 bit is enough). There were $O(s)$ such quantifiers. The final predicate verifies whether the transition from one configuration to another is valid, which takes $O(s)$ time since each configuration is $O(s)$ long. Thus the guessing stages of the machine take $O(s^2)$ time and the verification at the end takes $O(s)$ time, for a total running time of $O(s^2)$. □

Theorem 3. *ATIME*(t) $\subseteq$ *DSPACE*(t^2)

Proof. Let M be an alternating machine running in time t. If we separate M's execution into existential and universal stages, there are at most t many and each is at most t long. We begin by simulating M as long as it remains within the first stage. For the second stage of M, we simulate it for all possible choices from the first stage. There were at most t separate choices, so we can cycle through all of these using space $O(t)$. For the third stage of M, we must simulate it for all possible choices in the first and second stages. It takes space $O(t+t)$ to cycle through all of these. As there are a total of at most t stages, we can cycle through all possible guesses of M using $O(t^2)$ space. For the guesses corresponding to an existential stage, we ensure that at least one of the guesses results in an accepting computation; for the guesses corresponding to an universal stage, we ensure that all of the guesses result in an accepting computation.

Given a sequence of guesses, we simulate M with these guesses using the space-efficient universal Turing Machine from the first lecture. If we first convert M into an equivalent machine M' that uses $O(t)$ space, the total space usage is $O(t^2 + s_M) = O(t^2 + t) = O(t^2)$. □

Corollary 3. PSPACE $=AP$

Proof. ¿From Thm. 3, AP $\subseteq$ PSPACE. Also, PSPACE $\subseteq$ NPSPACE. Thus from Thm. 2, PSPACE $\subseteq$AP. $\square$

Theorem 4. *$ASPACE(s(n)) = \bigcup_{c>0} \mathrm{DTIME}(2^{c.s(n)})$. In other words, we need time exponential in the space requirement to deterministically simulate an alternating TM.*

Proving this theorem will be a homework problem.

Corollary 4. *$AL = \mathrm{P}$*

Corollary 5. *$APSPACE = \mathrm{EXP}$*

7 Next Lecture

Next time, we will use alternation to prove the non-existence of SAT-solvers that are both time and space efficient. The main topic of next class is non-uniformity where we allow different algorithms for inputs of different lengths.