MASSACHUSETTS INSTITUTE OF TECHNOLOGY
18.405J / 6.841J, ADVANCED COMPLEXITY THEORY

Descriptive Complexity and Some Unusual Quantifiers

Chelsea Voss

May 6, 2016

Abstract. It is possible to define $\forall$ and $\exists$ operators which operate on complexity classes to produce new definitions of complexity classes; this is one way to define the classes in the polynomial hierarchy. In this survey, I investigate what new complexity class operators I can define beyond the ones we discussed in 6.841. I define a complexity class HP using the Henkin quantifier H, and discuss my original analysis of its properties.

1 Introduction

It is possible to extend the definitions of the operators $\exists$ and $\forall$ in order to define operations on complexity classes. For any complexity class C, we can define $\exists\mathsf{C}$ and $\forall\mathsf{C}$ as follows: [3]

$$\exists \mathbf{C} \equiv \{L \mid \exists V : \text{Turing machine, such that } L(V) \in \mathbf{C} \text{ and } L = \{x \mid \exists w_1.\ V(w_1, x) \text{ accepts}\}.\}$$

$$\forall \mathbf{C} \equiv \{L \mid \exists V : \text{Turing machine, such that } L(V) \in \mathbf{C} \text{ and } L = \{x \mid \forall w_1.\ V(w_1, x) \text{ accepts}\}.\}$$

In these definitions, require that w_1 be a string whose length is at most polynomial in $|x|$. Then, for example, $\mathbf{NP} = \exists \mathbf{P}$: for every language in $\mathbf{NP}$, there exists a verifier V such that for some polynomial-length witness string w_1, $V(w_1, x)$ accepts. Similarly, $\mathbf{coNP} = \forall \mathbf{P}$.

Other operators can be defined over complexity classes, including the operators BP and $\oplus$. All four of these operators were developed during 6.841 as we discussed Toda's Theorem. Conveniently, $\mathsf{BP}\mathbf{P}$ is equivalent to $\mathbf{BPP}$. [3]

$$\mathsf{BP}\mathbf{C} \equiv \{L \mid \exists M : \text{Turing machine, such that } L(M) \in \mathbf{C} \text{ and } \Pr_{w_1}[M(w_1, x) \text{ accepts} \Leftrightarrow x \in L] \geq 2/3.\}$$

$$\oplus \mathbf{C} \equiv \{L \mid \exists M : \text{Turing machine, such that } L(M) \in \mathbf{C} \text{ and } L = \{x \mid \text{there are an odd number of } w_1\text{s such that } M(w_1, x) \text{ accepts}\}.\}$$

All of these operators are interesting because they allow concise definitions of $\mathbf{NP}$, $\mathbf{coNP}$, and $\mathbf{BPP}$ in terms of operators applied to $\mathbf{P}$. These operators also permit us to define each of the classes in the polynomial hierarchy in terms of $\exists$ and $\forall$: [3]

$$\begin{aligned} \Sigma_1^P &= \exists \mathbf{P} \\ \Pi_1^P &= \forall \mathbf{P} \\ \Sigma_2^P &= \exists \forall \mathbf{P} \\ \Pi_2^P &= \forall \exists \mathbf{P} \\ \Sigma_3^P &= \exists \forall \exists \mathbf{P} \\ &\cdots \end{aligned}$$

These operators are a fascinating abstraction, as they make it easier to define complicated complexity classes and intuit their relationships to other classes.

Indeed, on the way to proving Toda's Theorem we used these quantifiers in order to prove that $\mathsf{PH} \subseteq \mathsf{BP} \oplus \mathsf{P}$! [3]

The results of *descriptive complexity theory* make these relationships even more fascinating. This is a field seeking to characterize complexity classes according to which logical formalisms are sufficient to express each class's languages; beyond NP and coNP, descriptive complexity has characterized NL, PSPACE, EXPTIME, and others in terms of first-order or second-order logic with and without certain extra operators. [8] Descriptive complexity is also an interesting object of study because it does not require us to define a model of computation such as Turing machines; instead, the logics themselves fulfill this role.

Logical quantifiers therefore seem like a fruitful way to define new abstractions for complexity classes. The purpose of this survey will be to investigate what new complexity classes can be defined using operators based on as-yet-uninvestigated logical quantifiers. We will consider some candidate logical quantifiers; I will focus on discussing *branching quantifiers* and the complexity classes that they permit, and will analyze some of the properties of those complexity classes.

2 More operators

Here are the results of my attempts to construct new operators for complexity classes.

2.1 co

Definition 2.1. The co in such classes as coNP can itself be thought of as an operator that acts on complexity classes. The following operator definition captures the relationship between classes and co-classes more generally:

$$\mathsf{co}\mathbf{C} \quad \equiv \quad \{L \mid \overline{L} \in \mathbf{C}\}$$

The relationships of classes to their co-classes is in general extremely well-characterized; the primary interesting behavior of co as an operator is that it may be nested or rearranged among collections of other operators. For example, a parallel version of De Morgan's Laws holds for co, $\exists$, and $\forall$:

Definition 2.2 (De Morgan's Laws).

$$\neg\forall x.\phi(x) = \exists x.\neg\phi(x)$$

$$\neg\exists x.\phi(x) = \forall x.\neg\phi(x)$$

Theorem 2.1.

$$\mathsf{co}\forall\mathbf{C} = \exists\mathsf{co}\mathbf{C}$$

$$\mathsf{co}\exists\mathbf{C} = \forall\mathsf{co}\mathbf{C}$$

Proof. Simply unpack the definitions of $\exists$, $\forall$, and co, then apply De Morgan's Laws inside the definitions. For example, any language L of strings x such that there does not exist a witness w such that $V(w,x)$ accepts is also a language L of strings x such that for every w, $V(w,x)$ does not accept. □

2.2 $\exists!$

I encountered the $\exists!$ operator while searching for unusual logical quantifiers. This is the "there exists only one" quantifier. We can use it to define a complexity class operator in the same way we did with $\exists$:

Definition 2.3.

$$\begin{aligned}\exists!\mathbf{C} \quad \equiv \quad & \{L \mid \exists V : \text{Turing machine, such that } L(V) \in \mathbf{C} \text{ and} \\ & L = \{x \mid \exists! w_1.\ V(w_1, x) \text{ accepts}\}.\}\end{aligned}$$

We discussed *UniqueSAT* in 6.841 when covering the Valiant-Vazirani theorem; naturally, *UniqueSAT* is the complete problem for $\exists!\mathbf{P}$. [3] $\exists!\mathbf{P}$ happens to be a class that has already been defined and characterized: it is the same as the complexity class $\mathbf{US}$. [4] [6]

Finally, I'll mention a theorem about $\exists!\mathbf{P}$: $\exists!\mathbf{P}$ is contained within $\exists\forall\mathbf{P}$.

Theorem 2.2.

$$\exists!\mathbf{P} \subseteq \exists\forall\mathbf{P}$$

Proof. Every predicate $\exists!x.\phi(x)$ can be rewritten as an equivalent predicate of the form $\exists x.\forall y.\psi(x,y)$ as follows. Note that $\Rightarrow$ and $\neq$ can be defined as boolean operations. [1]

$$\begin{aligned}\exists!x.\,\phi(x) \quad &= \quad \exists x.\ x \text{ is the unique solution to } \phi \\ &\qquad \exists x.\,\phi(x) \text{ holds, and no other } y\ (y \neq x) \text{ satisfies } \phi \\ &\qquad \exists x.\forall y.\,\phi(x) \wedge ((y \neq x) \Rightarrow \neg\phi(y))\end{aligned}$$

□

2.3 H

The *Henkin quantifier* H is an example of a *branching quantifier*, a class of logical quantifiers in which the choice of variables along different branches is forced to be independent. H is defined as follows: [10] [9]

Definition 2.4 (Henkin quantifier). [7]

$$H(u,v,w,x).\,\phi(u,v,w,x) \equiv \begin{pmatrix}\forall u & \exists v \\ \forall w & \exists x\end{pmatrix}\phi(u,v,w,x)$$

This matrix of quantifiers behaves like $\forall u.\,\exists v.\,\forall w.\,\exists x.\,\phi(u,v,w,x)$, except for one crucial detail: the values of u and v must be chosen independently of the values of w and x. Curiously, this quantifier cannot be expressed using first-order logic. It can, however, be expressed in second-order logic; we discuss this later. [7]

Inspired by this, I define an operator H for complexity classes. Note that the witness strings $w_1 \ldots w_4$ must all be polynomial in length:

$$\begin{aligned}\mathsf{HC} \quad &\equiv \quad \{L \mid \exists V : \text{Turing machine, such that } L(V) \in \mathsf{C} \text{ and} \\ &\qquad L = \{x \mid \begin{pmatrix}\forall w_1 & \exists w_2 \\ \forall w_3 & \exists w_4\end{pmatrix} V(w_1,w_2,w_3,w_4,x) \text{ accepts}\}.\}\end{aligned}$$

2.3.1 Intuition: a complete problem for HP

For more intuition about H, let's try to figure out a complete problem for HP, something that is to HP like *SAT* is to NP.

First, a reminder about the complete problems for classes such as $\exists\mathsf{P}$ and $\exists\forall\exists\mathsf{P}$. For $\exists\mathsf{P}$, *SAT* is a complete problem: the ability to decide for some x

whether there exists some witness w such that polynomial-time $V(w,x)$ accepts (or, equivalently, such that some polynomially-sized circuit $\phi(w,x)$ evaluates to 1) is a problem that every language in $\exists\mathbf{P}$ can be reduced to.

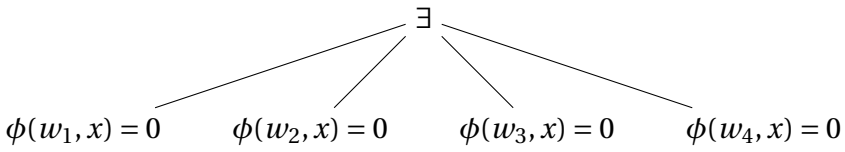


Figure 1: *SAT* as a one-level game tree. This one evaluates to 0.

For $\exists\forall\mathbf{P}$, we instead need to decide for some x whether there exists w_1 such that for all w_2, $V(w_1,w_2,x)$ accepts – this is the complete problem for $\exists\forall\mathbf{P}$. To make it a little less contrived, imagine a game tree of two layers: first the $\exists$ player chooses a move, then the $\forall$ player. At the end of the tree is a bit determining who has won, the $\exists$ player (if the bit is a 1) or the $\forall$ player (if the bit is a 0). Thus, the complete problem for $\exists\forall\mathbf{P}$ is the problem of deciding whether any perfectly-played general game tree of depth two results in a win for $\exists$ or a win for $\forall$. If you can decide this problem in polynomial time, you can decide any problem in $\exists\forall\mathbf{P}$ in polynomial time.

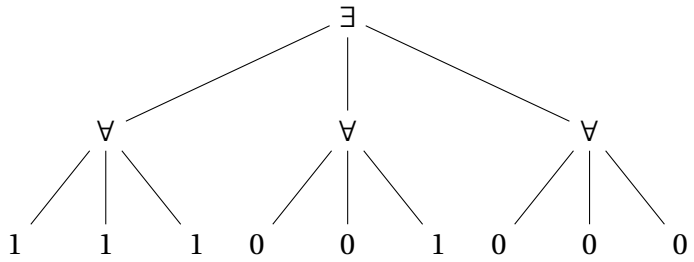
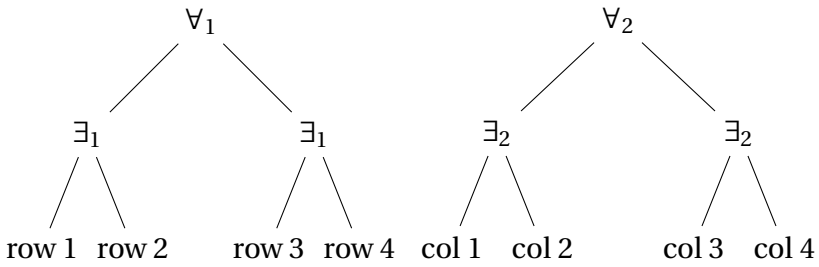


Figure 2: $\exists\forall\mathbf{P}$ problems reduce to the problem of finding the winner in a two-level game tree. $\exists$ can win this one by choosing the leftmost branch. At each point, the players are outputting a polynomially-long string, so they have exponentially many choices.

This intuition applies to the rest of the polynomial hierarchy: $\forall\exists\forall\mathbf{P}$ problems can be reduced to game trees of depth three where the $\forall$ player goes first instead of the $\exists$ player; $\exists\forall\exists\forall\exists\mathbf{P}$ problems can be reduced to game trees of depth five where the $\exists$ player goes first; and so on.

Back to HP. What is the corresponding HP-complete game? With the H quantifier, we have two game trees playing out in parallel – $\binom{\forall w_1\ \exists w_2}{\forall w_3\ \exists w_4}$. Imagine that the two games happen in separate rooms, Room 1 and Room 2, each with its own $\forall$ player and $\exists$ player.

In the traditional $\forall\exists\forall$-style game trees, the results of the game are stored as 1s and 0s in each of the leaves of the tree. Here, the result of the game will instead be a function of which leaf *each* game tree ends up with. Perhaps instead of placing 1s and 0s onto leaves, we place the 1s and 0s into a 2×2 matrix; the result of the game in Room 1 goes towards choosing a row of the matrix, and the result of the game in Room 2 goes towards choosing a column of the matrix. The values in the matrix are all determined by the choice of $\phi(w_1, w_2, w_3, w_4, x)$ that the game is being played over.



$\phi \rightarrow$

0	1	1	0
1	0	1	1
1	1	1	0
0	0	1	1

Figure 3: The four-player game for HP. Two game trees played separately, with the result of the overall game scored by the combined results of the two parallel games. This one is a guaranteed win for 1 (players $\exists_1$ and $\exists_2$).

The $\forall$ players are on the same team, trying to get the outcome of the game to be a 0; the $\exists$ players are similarly on the same team, trying to get the outcome of the game to be a 1. The players in separate rooms make their moves in turn, and then the final result of the game is calculated on the matrix. Like before, the players are choosing polynomially-long strings, so they have exponentially many moves to evaluate.

Thus, any problem in $\mathsf{H}\mathbf{P}$ can reduce to the problem of determining, given some ϕ, whether the game setup as described above necessarily evaluates to a win for 0 or a win for 1. Let's call this game *HSAT*. Since any language in $\mathsf{H}\mathbf{P}$ can be reduced to *HSAT*, *HSAT* is $\mathsf{H}\mathbf{P}$-hard; since *HSAT* $\in \mathsf{H}\mathbf{P}$, *HSAT* is $\mathsf{H}\mathbf{P}$-complete.

Definition 2.5 (HSAT). *HSAT* is a complete problem for $\mathsf{H}\mathbf{P}$, as described above.

2.3.2 Guaranteed winners?

We're not done yet: in order to treat this *HSAT* setup as a game, we have to check that it satisfies a certain property. One of the nice things about $\forall - \exists$ game trees is that they have a *guaranteed winner*. If the $\exists$ player is not guaranteed to win, then there exists a strategy by which the $\forall$ player can play such that no matter what the $\exists$ player does, the $\forall$ player will win. Vice versa, if the $\forall$ player is not guaranteed to win, then the $\exists$ player has a guaranteed winning strategy. [2]

Definition 2.6 (Guaranteed winner property).

$$\neg\, \exists\, \text{strategy}_{\text{p0}}.\ \forall\, \text{strategy}_{\text{p1}}.\ \text{p0 wins} \leftrightarrow \exists\, \text{strategy}_{\text{p1}}.\ \forall\, \text{strategy}_{\text{p0}}.\ \text{p1 wins}$$

We would prefer that *HSAT* games also have the guaranteed winner property, because if they do not, then we do not in all cases have a clear definition of which player is the unique winner of an *HSAT* game, and thus do not have a clear definition of whether the value of the game is 0 or 1.

In fact, I have found a counterexample that demonstrates that *HSAT* does *not* have this property. Consider the game below.

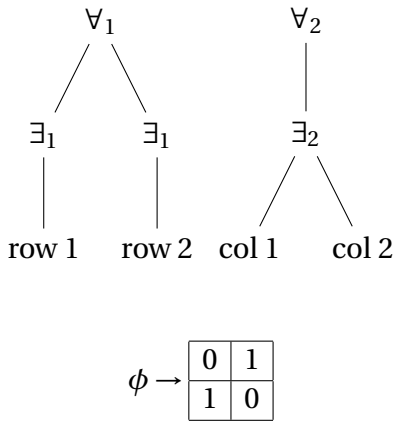


Figure 4: The ∀ players do not have a guaranteed win for this game: no matter what $\forall_1$ picks, there's a strategy for $\exists_2$ that defeats it. However, the ∃ players also do not have a guaranteed win for this game. No matter what $\exists_2$ picks, there's a strategy for $\forall_1$ that defeats it.

Fortunately, if there is a way to interpret the logical formula $H(\ldots)\phi(\ldots)$ as being true or false for every ϕ, then we can use that interpretation to define a unique winner of the *HSAT* game. I found an elaboration on the definition of H which clarifies this issue, stating that the Henkin quantifier is equivalent to the following: [5]

$$\begin{pmatrix} \forall x_1 & \exists y_1 \\ \forall x_2 & \exists y_2 \end{pmatrix} \phi(x_1, x_2, y_1, y_2) \equiv \exists f.\, \exists g.\, \forall x_1.\, \forall x_2.\, \phi(x_1, x_2, f(x_1), g(x_2))$$

This is a *skolemization*: it expresses H in second-order logic. Second-order logic allows us to quantify over functions instead of just values. Instead of having the ∃ players choose their y_1 and y_2, this definition has the ∃ players choose their strategies f and g in advance, as functions. Note that f only depends on x_1 and g only depends on x_2, so the two branches of the logic are independent of each other.

Thus, if there is no guaranteed-win for the ∀ players and also no guaranteed-win for the ∃ players, then we resolve the game in favor of the ∀ player. The *HSAT* game only has a value of 1 if the ∃ players have some strategy by which they can achieve a guaranteed win. Even though *HSAT* games don't have the guaranteed winner property, we now know how to define a winner for every game.

2.3.3 Known results

I could not find any prior results that deal with the same complexity class HP as defined above, via intentionally deriving a definition from H. Nor could I find any complexity classes which are unintentionally equivalent; I hunted through the literature on two-prover systems to double-check. However, I did find some results that investigate the complexity of queries which use the Henkin quantifier H, just not in exactly the way I'm interested in. [9]

One result is as follows: we prove that a weaker variant of H defines a complexity class that, though weaker, contains the class NP. [7]

Definition 2.7 (*WEAK-HSAT*). Call *WEAK-HSAT* the problem of deciding whether $\binom{\forall x_1\ \exists a}{\forall x_2\ \exists b}\phi(x_1, x_2, u, v)$ is satisfiable for a given ϕ, where x_1 and x_2 may be polynomially-long strings, *but* where a and b must have length $O(1)$.

Theorem 2.3. *WEAK-HSAT* is NP-hard.

Proof. The proof is by reduction from *3COLORING*, a known NP-hard problem, to *WEAK-HSAT*.

For any instance G of a graph that we wish to decide the 3-colorability of, let V be the vertices in G, and let $E(x, y)$ be a binary relationship which is true if x and y share an edge, or false otherwise. Let C be the set $\{0, 1, 2\}$, denoting three colors.

Then, $\binom{\forall x_1 \in V\ \exists a}{\forall x_2\ \exists b}\phi(x_1, x_2, a, b)$ is true if and only if the graph G is 3-colorable, for the following ϕ:

$$\phi = ((x_1 = x_2) \Rightarrow (a = b)) \wedge (E(x_1, x_2) \Rightarrow (a \neq b))$$

Note that x_1 and x_2 are vertex labels chosen from V, and a and b are colors chosen from C; thus, the truth of this formula can be computed by *WEAK-HSAT*.

The first clause of ϕ requires that for any two vertex labels which are identical, the two branches of logic must have independently decided on the same colors a and b, thus must color all of the nodes consistently. The second clause of ϕ requires that if the vertices x_1 and x_2 share an edge, then the two branches of logic must have independently decided on two different colors a and b. Overall, there exists a strategy which the $\exists$ players can use to satisfactorily choose the colors a and b iff the graph is 3-colorable. □

2.3.4 Results for HP

Here are the results of my investigations of HP.

Classes in HP. HP can decide any decision problem in ∀∃P simply by ignoring one of its game branches (more formal proof below). Open problem: whether any more powerful classes in the polynomial hierarchy are also contained within HP.

Theorem 2.4.

$$\forall\exists\mathsf{P} \subseteq \mathsf{HP}$$

Proof. To reduce any language $L \in \forall\exists\mathsf{P}$ to *HSAT*, I provide the following machine that decides L given a polytime *HSAT* decider. On input x,

1. Using the reduction from L to the complete problem for ∀∃P, convert x to some formula $\forall w_1 \exists w_2 \phi(w_1, w_2)$ that is true iff $x \in L$.

2. For the reduction, our putative *HSAT* solver takes in an input formula $\psi(z_1, z_2, z_3, z_4)$ over four variables, and accepts iff $\left(\begin{smallmatrix}\forall z_1 & \exists z_2 \\ \forall z_3 & \exists z_4\end{smallmatrix}\right) \psi(z_1, z_2, z_3, z_4)$.

3. Feed $\phi(w_1, w_2, \text{unused}, \text{unused})$ to this solver as input. This is a formula that ignores the variables z_3 and z_4; its truth is only determined by the values of z_1 and z_2. Intuitively, we have our two-team *HSAT* game, but only one team is playing; the other team's moves don't matter. (It is as if each row is all 1s or all 0s, such that the selection of column doesn't affect the outcome.)

4. Accept iff the *HSAT* solver accepts.

□

Classes containing HP. It is not obvious whether HP ⊆ ∀∃∀∃P. One might be tempted to trivially reduce an *HSAT* game to a ∀∃∀∃ game tree, but the analysis of a ∀∃∀∃ game tree might not necessarily match the outcome of the *HSAT* game: because the players have information about past moves during their two later moves, the outcomes may be different.

It is also not immediately obvious whether HP ⊆ PSPACE. In order to evaluate the *HSAT* game for some input ϕ, we can evaluate the skolemized formula: $\exists f.\ \exists g.\ \forall x_1.\ \forall x_2.\ \phi(x_1, x_2, f(x_1), g(x_2))$. A naïve approach would be to evaluate

every possible strategy f and g that the $\exists$ players could come up with. However, this doesn't work: f and g are functions from polynomial-length strings to polynomial-length strings; merely writing down the truth table of one of these functions requires exponentially many bits of space. In order to show that $\mathsf{HP} \subseteq \mathsf{PSPACE}$, we would need a cleverer approach, one that only uses polynomially many bits of space. I could not come up with an approach for PSPACE that provably decides the *HSAT* game correctly.

However, $\mathsf{EXPSPACE}$ clearly contains HP.

Theorem 2.5.

$$\mathsf{HP} \subseteq \mathsf{EXPSPACE}$$

Proof. We prove this by showing that the result of any *HSAT* game can be decided in $\mathsf{EXPSPACE}$. The input to a *HSAT* decision problem is some formula ϕ.

Exponential space is sufficient to allow us to keep track of one candidate tuple (f, g, x_1, x_2) at a time, and to increment up in order to search the entire space of candidates. We iterate over all of the candidates, searching for a candidate f and g such that for all choices of x_1 and x_2, $\phi(f, g, x_1, x_2)$ is true. □

Open problem: whether HP can be shown to be contained within any smaller complexity classes.

2.4 Other branching quantifiers

H is merely the tip of an iceberg: there are a rich array of branching quantifiers that can be defined. We can imagine quantifiers like H with more branches $(\begin{pmatrix} \forall u & \exists v \\ \forall w & \exists x \\ \forall y & \exists z \end{pmatrix})$ or with taller game trees in each branch $(\begin{pmatrix} \forall u & \exists v & \forall w \\ \forall x & \exists y & \forall z \end{pmatrix})$, as well as all of the varieties in between. I will not investigate these in depth, but it is worth noting that perhaps we could be able to use branching logic quantifiers to define a hierarchy similar to the polynomial hierarchy; because of the variety of ways to define bigger branching quantifiers, the structure of such a hierarchy would be complicated. I hypothesize that the entire hierarchy would be contained within $\mathsf{EXPSPACE}$.

3 Acknowledgments

Thanks to Alex Arkhipov for introducing the concept of the Henkin quantifier to me.

References

[1] Uniqueness quantification. https://en.wikipedia.org/wiki/Uniqueness_quantification. Accessed: 2016-05-06.

[2] 6.840 lecture notes, 2015.

[3] 6.841 lecture notes, 2016.

[4] Aaronson, Scott, Greg Kuperberg, and Christopher Granade. Complexity zoo. https://complexityzoo.uwaterloo.ca/Complexity_Zoo:U. Accessed: 2016-05-06.

[5] Badia, Antonio. Quantifiers in action: Generalized quantification in query, logical and natural languages. https://books.google.com/books?id=WC4pkt3m5b0C&pg=PA74&hl=en#v=onepage&q&f=false, 2009.

[6] Blass, A. and Y. Gurevich. On the unique satisfiability problem. *Information and Control* 55(1-3):80-88, 1982.

[7] Blass, Andreas, and Yuri Gurevich. Henkin quantifiers and complete problems. http://research.microsoft.com/en-us/um/people/gurevich/Opera/66.pdf, 1986.

[8] Immerman, Neil. Languages that capture complexity classes. https://people.cs.umass.edu/~immerman/pub/capture.pdf, 1987.

[9] Kolodziejczy, Leszek Aleksander. The expressive power of henkin quantifiers with dualization. http://www.mimuw.edu.pl/~lak/leszekmag.pdf, 2002.

[10] Sher, Gila. Ways of branching quantifiers. http://philosophyfaculty.ucsd.edu/faculty/gsher/bounds_of_logic_v_vi.pdf, 1991.

18.405J / 6.841J Advanced Complexity Theory

Spring 2016