COMS E6998: Proof Complexity and Applications (Spring'25) February 27, 2025

Lecture 5: Logic, Peano Arithmetic, and Bounded Arithmetic

Instructor: *Toniann Pitassi* Scribes: *Mark Chen, Saachi Mutreja*

1 Review of Logic

1.1 Basics and Definitions

We begin with a formal introduction of first order logic. Our starting building block is the notion of a language:

Definition 1. *A* ***language*** $\mathcal{L}$ *is specified by:*

1. *A set of function symbols, each with specified arity.*
2. *A set of predicate symbols, each of specified arity.*
3. *A set of variable symbols.*
4. *A set of logical symbols.*

Example 2. *For instance, we can consider the language* $\mathcal{L}_A$ *of arithmetic:*

1. *The function symbols are* $\{0, S, +, \cdot\}$*, where* 0 *has arity* 0*,* S *has arity* 1*,* $+$ *has arity* 2*, and* $\cdot$ *has arity* 2*.*
2. *The set of predicate symbols includes* $\{<, \leq, =\}$*. Here, each predicate has arity* 2*.*
3. *The variable symbols are arbitrary, but can include* $x, y, z, \ldots, a, b, c$ *and so on.*
4. *The logical symbols include* $\forall, \exists, \wedge, \vee, \neg$*.*

Next, we introduce the notion of *terms* over $\mathcal{L}$:

Definition 3. *A* term *over a language* $\mathcal{L}$ *is defined by inductively composing function symbols.*

Example 4. *We can consider simple examples of terms over the language* $\mathcal{L}_A$*:*

1. $+ \quad S0 \quad SSS0$ *corresponds with adding together the numbers* 1 *and* 3*, and thus evaluates to* 4*. Note that we will sometimes write this as* $S0 \quad + \quad SSS0$ *for readability.*
2. $(x + (SSy \cdot SS0)) \cdot Sx$*, corresponds to* $(x + 2(y + 2)) \cdot (x + 1)$*.*

Now, having defined terms over a language $\mathcal{L}$, we can introduce first order formulas:

Definition 5. *Given a predicate* P *of arity* k *from* $\mathcal{L}$ *applied to terms* $t_1, \ldots t_k$*, we say that the resulting expression* $P(t_1, \ldots t_k)$ *is an* ***atomic formula*** *of* $\mathcal{L}$*.*

We can also define $\mathcal{L}$***-formulas****: given* A, B *that are both* $\mathcal{L}$*-formulas, any application of logical symbols to this formulas is still an* $\mathcal{L}$*-formula. For instance,* $\neg A, A \vee B, A \wedge B, \forall x A$ *and* $\exists x B$ *are all still* $\mathcal{L}$*-formulas.*

Next, we introduce structures:

Definition 6. *An* $\mathcal{L}$***-structure*** $\mathcal{M}$ *consists of:*

1. *A non-empty set M called the underlying universe.*
2. *For every k-ary function symbol f, an associated k-ary function $f^{\mathcal{M}} : M^k \to M$.*
3. *For every k-ary relation symbol R, an associated k-ary relation $R^{\mathcal{M}} : M^k \to \{0,1\}$.*

Definition 7. *A variable x in a formula is called **free** if it is not quantified, and otherwise is called **bound**. A formula A is a **sentence** if all variables in A are quantified.*

Example 8. *We can consider the formula $\forall x, \exists y A(x, y, a)$. In this example, x, y are bound variables, and a is a free variable.*

In general, it is an accepted convention to let x, y, z denote bound variables, while a, b, c denote free variables.

1.2 Evaluating Sentences

Now, given a model $\mathcal{M}$, we can evaluate a given sentence A as being either true (1) or false (0).

Definition 9. *We say $\mathcal{M} \models A$, if A evaluates to true under $\mathcal{M}$.*

Example 10. *We can derive the following facts about when a sentence evaluates to true under a given model $\mathcal{M}$:*
1. *$\mathcal{M} \models P(t_1, \ldots t_k)$ if $P^{\mathcal{M}}(t_1, \ldots t_k) = 1$.*
2. *$\mathcal{M} \models \neg A$ if $\mathcal{M} \not\models A$.*
3. *$\mathcal{M} \models A \vee B$ if either $\mathcal{M} \models A$ or $\mathcal{M} \models B$.*
4. *$\mathcal{M} \models A \wedge B$ if both $\mathcal{M} \models A$ and $\mathcal{M} \models B$.*
5. *We say that $\mathcal{M} \models \forall x A(x)$ if $\forall m \in M$, $\mathcal{M} \models A(m/x)$.*
6. *We say that $\mathcal{M} \models \exists x A(x)$ if $\exists m \in M$, $\mathcal{M} \models A(m/x)$.*

Definition 11. *We say that $\models A$ (in words, A **is valid**), if and only if, for every model $\mathcal{M}$, $\mathcal{M} \models A$.*

Example 12. *For instance, let us consider the model $\mathcal{M}$ with underlying universe $\mathbb{N}$ and the usual definitions of $+, \cdot, s, 0$. Then,*
1. *$\mathcal{M} \models S0 + SS0 \leq SSS0$, but $S0 + SS0 \leq SSS0$ is not valid.*
2. *$\mathcal{M} \models \forall x \exists y (x + x = y)$, but again, $\forall x \exists y (x + x = y)$ is not valid.*

1.3 The LK System

Now, we are ready to introduce the LK system. A proof consists of lines:

Definition 13. *Lines of the proof are **sequents***

$$A_1, \ldots A_k \to B_1, \ldots B_\ell,$$

where the intended meaning is that $A_1 \wedge \cdots \wedge A_k \supset B_1 \vee \cdots \vee B_\ell$.

Definition 14. *LK has the following set of rules:*
1. ***Weakening:***

$$\frac{\Gamma \to \Delta}{\Gamma, A \to \Delta} \qquad \frac{\Gamma \to \Delta}{\Gamma \to \Delta, B}$$

2. ***Exchange:***

$$\frac{\Gamma, A, B, \Gamma' \to \Delta}{\Gamma, B, A, \Gamma' \to \Delta} \qquad \frac{\Gamma \to \Delta, A, B, \Delta'}{\Gamma \to \Delta, B, A, \Delta'}$$

3. ***Contraction:***

$$\frac{\Gamma, A, A \to \Delta}{\Gamma, A \to \Delta} \qquad \frac{\Gamma \to \Delta, A, A}{\Gamma \to \Delta, A}$$

4. ***Negation:***

$$\frac{\Gamma \to \Delta, A}{\Gamma, \neg A \to \Delta} \qquad \frac{\Gamma, A \to \Delta}{\Gamma \to \Delta, \neg A}$$

5. ***AND:***

$$\frac{A, B, \Gamma \to \Delta}{A \land B, \Gamma \to \Delta} \qquad \frac{\Gamma \to \Delta, A \quad \Gamma \to \Delta, B}{\Gamma \to \Delta, A \land B}$$

6. ***OR:***

$$\frac{A, \Gamma \to \Delta \quad B, \Gamma \to \Delta}{A \lor B, \Gamma \to \Delta} \qquad \frac{\Gamma \to \Delta, A, B}{\Gamma \to \Delta, A \lor B}$$

7. $\forall$*:*

$$\frac{A(t), \Gamma \to \Delta}{\forall x A(x), \Gamma \to \Delta} \qquad \frac{\Gamma \to \Delta, A(b)}{\Gamma \to \Delta, \forall x A(x)}$$

Here, we have used the notion that b is a free variable that only appears in A.

8. $\exists$*:*

$$\frac{A(b), \Gamma \to \Delta}{\exists x A(x), \Gamma \to \Delta} \qquad \frac{\Gamma \to \Delta, A(t)}{\Gamma \to \Delta, \exists x A(x)}$$

9. ***Axiom:***

$$A \to A$$

10. ***Cut rule:***

$$\frac{\Gamma, A \to \Delta \quad \Gamma \to \Delta, A}{\Gamma \to \Delta}$$

1.4 Soundness and Completeness of the System

Finally, we can introduce the notions of soundness and completeness. To start, we need the definition of when a sequent is valid:

Definition 15. *We say that a first-order sequent $A_1, \dots A_k \to B_1, \dots B_\ell$ is **valid** if and only if*

$$\models \neg(A_1 \land \cdots \land A_k) \lor (B_1 \lor \cdots \lor B_\ell).$$

With this, we can state the following theorem which establishes our ability to use LK to prove true statements:

Theorem 16. *1. **Soundness:** If a sequent has an LK proof, then it is valid.*

*2. **Completeness:** Every valid sequent has an LK proof.*

2 Peano Arithmetic

Definition 17 ($\mathcal{L}_{\text{PA}}$)**.** *In the language of arithmetic $L_{PA} = \{0, 1, +, \cdot, <, =\}$, the Peano arithmetic, is axiomatized by Robinson's arithmetic Q:*

1. $a + 1 \neq 0$.
2. $a + 1 = b + 1 \to a = b$.
3. $a + 0 = a$.
4. $a + (b + 1) = (a + b) + 1$.
5. $a \cdot 0 = 0$.
6. $a \cdot (b + 1) = (a \cdot b) + a$.
7. $a \neq 0 \to \exists x, x + 1 = a$.

Importantly, L_{PA} has all of the rules / axioms of LK, in addition to the *induction rule*:

Definition 18 (Induction Rule)**.** *The induction rule states that*

$$\frac{A(b), \Gamma \to \Delta, A(b+1)}{A(0), \Gamma \to \Delta, \forall x A(x)}.$$

Here, we use "1" as an abbreviation for $S0$.

3 Bounded Arithmetic

Bounded arithmetic is a collection of weaker sub-theories of Peano arithmetic, in that quantifiers are bounded in the induction axiom or equivalence postulates (as we'll soon seen, such bounds arise as part of the quantifiers, e.g. $\forall x \leq t$, etc). Bounded arithmetic is restricted enough that it is related to constructive proof systems:

Constructive Proof System. A proof system is called *constructive* if every proof of an existential statement implicitly (or explicitly) contains an algorithm for finding a witness. In other words, whenever the system proves a formula of the form

$$\forall x \, \exists y \quad \varphi(x, y),$$

the proof must encode a method (or procedure) by which one can compute, for every input x, an appropriate y such that $\varphi(x, y)$ holds. This means that the proof itself is not merely nonconstructive or existential; it gives rise to a way of "**witnessing**" the existential quantifier.

Feasibly Constructive Proof System A *feasibly constructive* proof system is a constructive proof system with the additional requirement that the witness extraction is **efficient**. That is, not only does every proof of an existential statement yield an algorithm for finding a witness, but this algorithm is computationally feasible (typically, it runs in polynomial time).

Remark 19. *These proof systems capture the idea that, if "$\forall x \exists y \, A(x, y)$" is provable then there should be a feasible algorithm to find as a function of x. As will be introduced next, S_2^1 will be a feasible proof* *system, and S_2^i and T_2^i are proof systems that have proof theoretic strength that corresponds to higher levels of the polynomial time hierarchy (connection discussed in section 3.4).*

3.1 Motivation

With the discussion about constructive proof systems in mind, bounded arithmetic had the motivation to reflect feasible (i.e., efficiently verifiable) reasoning by limiting the induction and quantification to bounded formulas. Although these restrictions make the induction principle and quantifiers weaker, the definition needs to be designed so that certain operations also make it stronger to express all polynomial-time computable functions. The "smash" operation (denoted by #) in particular is used to obtain a polynomial growth rate in the following sense: suppose you have $a = a_1, \ldots, a_k$ as the k free variables to your formula, and so $t(a)$ is a number whose length in binary is polynomial in the sum of lengths of the input numbers for any term $t(x_1, \ldots, x_k)$ built up using the "smash" operation with other built-in functions $(S, +, \cdot$, etc.).

In summary, the design of bounded arithmetic reflects the intuition that a "good" proof of an existence statement should contain a feasible method for finding the object whose existence is claimed.

3.2 Language of bounded arithmetic

We first define the language of bounded arithmetic (denoted $\mathcal{L}_{\mathrm{BA}}$) by extending $\mathcal{L}_{\mathrm{PA}}$ in definition 17 by the following new samples:

1. Additional function symbols:
   - $|x|$, which is the length of $x \in \mathbb{N}$ when written in binary representation.
   - $\lfloor x/2 \rfloor$, which has the standard meaning.
   - $x\#y$, which has arity of 2 and has the following meaning:

$$x\#y = 2^{|x|\cdot|y|}.$$

Remark 20. *Not only is the # helpful to get the right growth rate for polynomial time computation, it also ensures that the* Quantifier Exchange Principle *holds:*

$$(\forall x \leq |a|)(\exists y \leq b)\ A(x,y) \leftrightarrow (\exists y \leq (2a+1)\#(4(2b+1)^2))(\forall x \leq |a|)\ [A(x, \beta(x+1,y)) \wedge \beta(x+1,y) \leq b]$$

2. Relation symbols remain the same as $\mathcal{L}_{\mathrm{PA}}$.
3. Additional logical symbols:
   - Bounded quantifiers: quantifiers like $\forall x \leq t\ A(x,a)$, and $\exists x \leq t\ A(x,a)$, where x runs over all natural numbers up to $\mathbf{poly}(|a|)$.
   - Sharply bounded quantifiers: quantifiers like $\forall x \leq |t|\ A(x,a)$, and $\exists x \leq |t|\ A(x,a)$, where x runs over all natural numbers of lengths up to $\mathbf{poly}(|a|)$.

3.3 Bounded arithmetic proofs

Then, lines of proofs are sequents over $\mathcal{L}_{\mathrm{BA}}$ as defined in the previous section.

Remark 21. *Since the quantifiers and the # operation enable proofs that may not be efficiently computed, the induction rules are also designed to restrict the quantifiers so that the proof can still be efficiently checked.*

As such, we add to all the rules and axioms of LK sequent calculus (see definition 14) the following:

1. It consists of the following axioms over all the operations (there are several such sets of axioms that are equivalent to each other, but we include the one from [**?**]'s thesis, where it is called the BASIC axioms):

   (1) $y \leq x \to y \leq Sx$
   (2) $x \neq Sx$
   (3) $0 \leq x$
   (4) $x \leq y \wedge x \neq y \leftrightarrow Sx \leq y$
   (5) $x \neq 0 \to 2 \cdot x \neq 0$.
   (6) $y \leq x \vee x \leq y$
   (7) $x \leq y \wedge y \leq x \to x = y$
   (8) $x \leq y \wedge y \leq z \to x \leq z$
   (9) $|0| = 0$
   (10) $x \neq 0 \to |2 \cdot x| = S(|x|) \wedge |S(2 \cdot x)| = S(|x|)$
   (11) $|S0| = S0$.
   (12) $x \leq y \to |x| \leq |y|$
   (13) $|x\#y| = S(|x| \cdot |y|)$
   (14) $0\#y = S0$
   (15) $x \neq 0 \to 1\#(2 \cdot x) = 2(1\#x) \wedge 1\#(S(2 \cdot x)) = 2(1\#x)$
   (16) $x\#y = y\#x$
   (17) $|x| = |y| \to x\#z = y\#z$
   (18) $|x| = |u| + |v| \to x\#y = (u\#y) \cdot (v\#y)$
   (19) $x \leq x + y$
   (20) $x \leq y \wedge x \neq y \to S(2 \cdot x) \leq 2 \cdot y \wedge S(2 \cdot x) \neq 2 \cdot y$
   (21) $x + y = y + x$
   (22) $x + 0 = x$
   (23) $x + Sy = S(x + y)$
   (24) $(x + y) + z = x + (y + z)$
   (25) $x + y \leq x + z \leftrightarrow y \leq z$
   (26) $x \cdot 0 = 0$
   (27) $x \cdot (Sy) = (x \cdot y) + z$
   (28) $x \cdot y = y \cdot x$
   (29) $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
   (30) $x \geq S0 \to (x \cdot y \leq x \cdot z \leftrightarrow y \leq z)$
   (31) $x \neq 0 \to |x| = S(||\lfloor x/2 \rfloor||)$
   (32) $x = \lfloor y/2 \rfloor \leftrightarrow (2 \cdot x = y \vee S(2 \cdot x) = y)$

2. Additional rules:

   - Rules for bounded quantifiers: Recall that a sequent is usually written as $\Gamma \vdash \Delta$, where left side (the side of Γ) and right side (the side of Δ) are treated differently. So, for the two bounded quantifiers operations that were introduced, $\forall \cdot \leq \cdot$ and $\exists \cdot \leq \cdot$, we specify left side and right side rules for them respective (in propositional logic, we make s the concrete bound, and b and t the candidates for the quantified variables):

– ($\exists \leq$-left rule): If you can show for a concrete candidate b where the condition $b \leq s$ holds such that $A(b)$ is true, together with the rest of the assumptions, then it implies the corresponding bounded quantifier version:

$$\frac{b \leq s, A(b), \Gamma \to \Delta}{\exists x \leq s, A(x), \Gamma \to \Delta}.$$

– ($\exists \leq$-right rule): If you can prove $A(t)$ for some value t under the context Γ leading to Δ, then, with $t \leq s$ established, we can conclude the corresponding bounded arithmetic proof sequent:

$$\frac{\Gamma \to \Delta, A(t)}{t \leq s, \Gamma \to \Delta, \exists x \leq s\ A(x)}.$$

The ways that the following two bounded quantifiers rules are defined are analogous to the previous two, so we just write the rules down without paraphrasing them:

– ($\forall \leq$-left rule):

$$\frac{A(t), \Gamma \to \Delta}{t \leq s, \forall x \leq s\ A(x), \Gamma \to \Delta}.$$

– ($\forall \leq$-right rule):

$$\frac{b \leq s, \Gamma \to \Delta, A(b)}{\Gamma \to \Delta, \forall x \leq s\ A(x)}.$$

3. Restricted induction rule: Only formulas in which all quantifiers are bounded are allowed in the induction scheme.

- T_2^i are defined as the ones to have Σ_i^b-IND:

$$\frac{A(b), \Gamma \to \Delta, A(b+1)}{A(0), \Gamma \to \Delta, A(t)},$$

where "Σ_i^b" is because $A \in \Sigma_i^b$.

- S_2^i are defined as the ones to have Σ_i^b-PIND (P for polynomial):

$$\frac{A(\lfloor b/2 \rfloor), \Gamma \to \Delta, A(b)}{A(0), \Gamma \to \Delta, A(t)}$$

where "Σ_i^b" is because $A \in \Sigma_i^b$.
Note that, as it jumps from $A(\lfloor b/2 \rfloor)$ to $A(b)$, it is effectively doubling the size of the number in each induction step. Thus, to reach a number t from 0, it only takes $\log(t)$ induction steps, which is $\mathbf{poly}(|t|)$. Using the same reasoning, one would be able to give a directly easily verifiable proof in "$|t|$" following the IND-rules.

Remark 22. *Proof systems that use the PIND-rule is weaker than proof systems that use the IND-rule simply because the former assume a strictly stronger induction rule.*

Definition 23 (Σ_i^b and Π_i^b)**.** *For those who are familiar with the polynomial hierarchy, Σ_i^P denotes i alternating quantifiers starting with "$\exists$" where P defines a class of $\underline{p}$redicates. This is similar to Σ_i^b, where b stands for $\underline{b}$ounded and defines a restricted class of S_2^1* formulas. *In particular, when it says "bounded quantifiers", it means*

$$\Sigma_i^b : \underbrace{\exists x_1 \leq t_1(a), \forall x_2 \leq t_2(a), \dots}_{i \text{ alternations}} A(a, x_1, \dots, x_k),$$

and

$$\Pi_i^b : \underbrace{\forall x_1 \leq t_1(a), \exists x_2 \leq t_2(a), \dots}_{i \text{ alternations}} A(a, x_1, \dots, x_k),$$

where $A(\cdot)$ is a sharply bounded formula (sharply bounded formulas are well-defined syntactic objects that are poly-time computable, but there are poly-time computable formulas that cannot be written as a sharply bounded S_2^1 formula), and $t_i(\cdot)$ are built up from the built-in function symbols and the free variables in the formula, i.e. "a" in this particular case, though it can be any number of free variables in the formula. Similarly, when it says, "sharply bounded quantifiers", it means it means

$$\Sigma_i^b : \underbrace{\exists x_1 \leq |t_1(a)|, \forall x_2 \leq |t_2(a)|, \dots}_{i \text{ alternations}} A(a, x_1, \dots, x_k),$$

and

$$\Pi_i^b : \underbrace{\forall x_1 \leq |t_1(a)|, \exists x_2 \leq |t_2(a)|, \dots}_{i \text{ alternations}} A(a, x_1, \dots, x_k).$$

Remark 24. *We can think of Σ_k^b and Π_k^b for different levels of k to be related in the following way:*

1. ***(Base).*** *$\Sigma_0^b = \Pi_0^b$ is the set of formulas with only sharply bounded quantifiers.*
2. *If $A \in \Sigma_k^b$, then $(\forall x \leq |t|)A$ and $(\exists x \leq t)A$ are in Σ_k^b and $(\forall x \leq t)A$ is in Π_{k+1}^b. Conversely, if $A \in \Pi_k^b$, then $(\exists x \leq |t|)A$ and $(\forall x \leq t)A$ are in Π_k^b and $(\exists x \leq t)A$ is in Σ_{k+1}^b.*

3.3.1 Σ_k^p vs Σ_k^b

A quick note on the difference between Σ_k^p and Σ_k^b. The former defines a class of predicates and the latter defines a class of formulas. The strength of the two are related in the following way:

Theorem 25 ([?])**.** *Fix $k \geq 1$. A predicate Q is in $\Sigma_k^p \iff$ there is a Σ_k^b formula which defines it.*

Remark 26. *This theorem defines Σ, Π classes of the polynomial hierarchy syntactically without computation.*

3.4 Polynomial time "hierarchy" in proof complexity

In definition 23, we have mentioned Σ_i^b (resp. Π_i^b) is defined over semantically well-defined formulas. When it comes to Σ_1^p, where the p stands for a class of $\underline{p}$redicates, we can define the following symbols for the poly-time hierarchy:

$$\square_1^p = \mathbf{FP}, \Delta_1^p = \mathbf{P},$$

where **FP** means the poly-time computable functions, and **P** means the classic decision problems decidable by poly-time algorithms.

Definition 27 ($PB\exists(\Psi)$)**.** *Let Ψ be a set of predicates, p be a polynomial, and $B \in \Psi$. Then, $PB\exists(\Psi)$ means*

$$\vec{x} \in A \iff (\exists y \leq 2^{p(|\vec{x}|)})B(\vec{x}, y).$$

$PB\forall(\Psi)$ is analogously defined by replacing "$\exists$" with "$\forall$".

Thus, we can define the hierarchy as follows:

$$\Sigma_k^p = PB\exists(\Delta_k^p), \quad \Pi_k^p = PB\forall(\Delta_k^p).$$

As such, we have

$$\square_{k+1}^p = \mathbf{FP}^{\Sigma_k^p} = \mathbf{FP}^{\Pi_k^p}, \quad \Delta_{k+1}^p = \mathbf{P}^{\Sigma_k^p} = \mathbf{P}^{\Pi_k^p}$$

Example 28. $\Pi_1^p = \mathbf{coNP}$ *and* $\Sigma_1^p = \mathbf{NP}$*.*

This typical formulation of polynomial hierarchy is said to be purely logical characterizations of the classes in the poly-time hierarchy in terms of *expressibility* in a formal language. The goal of bounded arithmetic, in part, is to give purely logical characterizations of these same classes in terms of *derivability* in a formal theory.

To make it concrete what "derivability" and "formal theory" mean, we first define feasibility (as compared to the typical "efficiency") by formulas that are definable in different levels of Σ_i^b.

Definition 29 (Σ_i^b-definable)**.** *Let $f : \mathbb{N}^k \to \mathbb{N}$. The function f is called Σ_i^b-definable by a theory $R \iff$ there is a formula $A(\vec{x}, y) \in \Sigma_i^b$ such that*

1. *An acceptable witness y can be computed from input $\vec{x}$ by f: $\forall \vec{n} \in \mathbb{N}^k, A(\vec{n}, f(\vec{n})) = 1$.*
2. *Existence: $R \vdash (\forall \vec{x})(\exists y)A(\vec{x}, y)$*
3. *Uniqueness: $R \vdash (\forall \vec{x}, y, z)(A(\vec{x}, y) \wedge A(\vec{x}, z) \to y = z)$*

Example 30. *If a function is Σ_1^b-definable by a theory R, then there should be a Turing machine M which computes the function so that R can prove the Turing machine always halts within polynomial time. We call this function* provably recursive *in R. Concretely, S_2^1, which was defined to be the bounded arithmetic proofs that allow Σ_1^b-PIND rules, is a theory by which every poly-time function is Σ_1^b-definable. Furthermore, every poly-time predicate is Δ_1^b-definable by S_2^1.*

4 Witnessing theorems

**For a more detailed survey on these results, see sections 2 - 4 of [?]. We will also point out specific subsections for specific results below.*

We have described, roughly, how formal theories provide different characterizations of the polynomial hierarchy and what roles S_2^i's of different i's play in these alternative characterizations. Witnessing theorems are meant to give an exact characterization of the Σ_i^b-definable functions of S_2^i in terms of computational complexity.

Theorem 31 (For S_2^1)**.** *Let $A := \forall a\, \exists x \leq t(a)\ B(a,x)$ be a Σ_1^b-definable formula in S_2^1. Then there is a function $f \in \square_1^p = \mathbf{FP}$, such that*

$$\mathbb{N} \vDash \forall a\, B(a, f(a)).$$

We can generalize the above theorem to S_2^i and T_2^i:

Theorem 32 (For S_2^i)**.** *Let $i \geq 1$. Let $A := \forall a\, \exists x \leq t(a)\ B(a,x)$ be a Σ_i^b-definable formula in S_2^i. Then there is a function $f \in \square_i^p$ and a bound t, such that*

- $S_2^i \vdash (\forall a, b)(B(a,b) \to A(a,b))$
- $S_2^i \vdash (\forall a)(\exists! b)(B(a,b))$
- $S_2^i \vdash (\forall a)(\exists b \leq t(a))(B(a,b))$
- $\mathbb{N} \vDash B(n, f(n))$*, for all* n*.*

Proof. We show an outline of this theorem, and theorem 31 follows as a corollary by setting $i = 1$.

We first introduce the idea of free-cut elimination.

Definition 33 (Free cut inference)**.** *A cut inference*

$$\frac{\Gamma \to \Delta, A \qquad A, \Pi \to \Lambda}{\Gamma, \Pi \to \Delta, \Lambda}$$

is free unless A is the direct descendant of either of a formula in an initial sequent or of a principal formula of an induction inference (for what these mean, review sequent calculus).

Theorem 34 (Free-cut elimination)**.** *For S_2^i-proof (resp. T_2^i-proof) P, there exists proof P^* in the same theory with the same endsequent which contains no free cut.*

Here are the steps to prove the witnessing theorem for S_2^i:

1. Given S_2^i-proof P such that, fixing any $\vec{x}$, $P \to (\exists y)A(\vec{x}, y)$ is true. Then, by theorem 34, there is a proof P^* in S_2^i such that

   $$P^* \to (\exists y \leq t(\vec{x}))\ A(\vec{x}, y)$$

   where every formula in P^* is a Σ_i^b-formula.

2. Based on P^*, extract an algorithm to compute $f(\vec{x})$ such that $A(\vec{n}, f(\vec{n}))$ evaluates to true for all $\vec{n}$. It needs to be shown that

   - $f \in \square_i^p$,
   - f is Σ_i^b-definable by S_2^i, and
   - $S_2^i \vDash (\forall \vec{x})\ A(\vec{x}, f(\vec{x}))$.

   In other words, P^* serves as a program (as well as a proof) that satisfies the conditions in the theorem.

See sections 3.5 - 3.7 of [**?**] for a full proof of the second step (first step follows directly from theorem 34). $\square$

The witnessing theorem for T_2^i is related to **PLS** defined in [**?**]. It will be a digression of this lecture's presentation, so let's refer to section 4.2 of [**?**] for a more detailed discussion (it suffices to know that **PLS** is a class of functions conjectured to be strictly between $\square_1^p$ and $\square_2^p$). Here we only state the witnessing theorem concerning T_2^1.

Theorem 35 (For T_2^1 [**?**])**.** *Let A be a Σ_1^b-definable predicate, then there is a $f \in \mathbf{PLS}$, where $f(x) = y$, and a poly-time computable function π such that*

$$T_2^1 \vdash (\forall x) A(x, \pi \circ f(x)).$$

Furthermore, every **PLS** *function and π composing that function is Σ_1^b-definable by T_2^1.*

5 Propositional translations

For a more detailed survey on these results, see section 6 of [?**]. We will also point out specific subsections for specific results below.*

Finally, we see two important translations of proofs in bounded arithmetic into proofs in propositional logic, one from S_2^1 proofs to EF, and another from relativized $S_2^i(R)$ proofs to AC_i^0-Frege proofs.

5.1 From S_2^1 proofs to EF

One can think of S_2^1 as a uniform proof system, whereas EF is the corresponding non-uniform proof system. The translation goes roughly as follows:

For any Σ_1^b-definable formula $A := \forall a\, \exists x \leq t(a)\ B(a, x)$ in S_2^1, there is a sequence of propositional statements $[[A]]_n$ (expressing A for $m \in \mathbb{N}$, $|m| = n$) such that $[[A]]_n$ has a poly-size EF proofs. This conversion is done by free cut elimination and prof. Sam Buss described in a lot more details in section 6.1 of [**?**] with implications of this translation in section 6.2.

5.2 From relativized $S_2^i(R)$ proofs to AC_i^0-Frege proofs [**?**]

It is helpful to think of Σ_i^b formula of $S_2^i(R)$ as corresponding to a predicate in i-th level of relativized polynomial hierarchy (**i.e. analogous to the translation from a relativized polynomial hierarchy to bounded depth circuits**). The translation goes as follows: for any Σ_i^b-definable formula A^R provable in $S_2^i(R)$, there is a sequence of propositional statements $[[A^R]]_n$ (expressing A^R for all $m \in \mathbb{N}$, $|m| = n$) such that $[[A^R]]_n$ has quasi-poly size $\mathbf{AC}_i^0$-Frege proofs.

Example 36. $\mathsf{PHP}(R)$ = *pigeonhole principle for relation R, which can be thought of as a binary relation that is like what we have seen in a previous lecture ($R(i, j)$ indicates if pigeon i is in hole j). Then, this R asserts that*

$$\forall a \Big[\Big(\exists x \leq a+1\, \forall y \leq a \quad \neg R(x, y) \Big) \vee \Big(\exists x_1, x_2 \leq a+1\, \exists y \leq a \quad (x_1 \neq x_2 \wedge R(x, y) \wedge R(x_2, y)) \Big) \Big].$$

Then, the propositional transition is $[[\mathsf{PHP}^R]]_n$, where we set $a := n, n \in \mathbb{N}$, and the propositional variables *are $R_{i,j}$ for $i = n+1, j \leq n$ such that*

$$\bigvee_{i\in[n+1]} \bigwedge_{j\in[n]} \neg R_{i,j} \vee \bigvee_{\substack{i_1\neq i_2 \\ i_1,i_2\in[n+1]}} \bigvee_{j\in[n]} R_{i_1,j} \wedge R_{i_2,j}.$$

With this new three-relationship with R to express the PHP*, we could quantify to prove this in S_2^d, as all the other terms are bounded, and this is a Σ_2^d formula with just two levels of alternation. To build a circuit, the rough idea is that for every fixed a, we have a propositional formula (relation R where you have a holes for $b+1$ pigeons).*

There are three main steps to translate $S_2^i(R)$ to $\mathbf{AC}^0$-Frege:

1. **(Free cut-free elimination).** Use cut elimination to show that if "$\to \forall a \exists x \leq t(a)\ B(a,x)$" has a $S_2^i(R)$ proof, then there is another $S_2^i(R)$ proof Π of "$\to A$" where all formulas in the proof are in $\Sigma_i^b(R)$.

   Let $S_2^i(R) \vdash A^R(a)$ $A \in \Sigma_i^b(R)$. Then, there is an $S_2^i(R)$ proof Π of $A^R(a)$ with no "free cuts" (recall theorem 34 and the definition before it). Also, we can assume that the only free variables in the proof are those occurring in an induction inference, plus a, the free variable in A.

2. Fix $n \in \mathbb{N}$, i.e. consider the restricted statement "$\to \exists x \leq t(n)\ B(a,x)$". Then, translate each *line* in Π to an $\mathbf{AC}^0$-formula.

   Suppose R is a binary relation, so $k = 2$ (WLOG, as the idea is analogous for all values of k). Let $A(a) \in \Sigma_i^b(R)$. The corresponding propositional variables are $r_{i,j}$ for $i,j \in \mathbb{N}$. Then, we define inductively $[[A(n)]]$:

   (a) If $A(a)$ is quantifier-free, and does not contain R, then $[[A(n)]] = 1$ if $A^R(n)$ is valid, 0 otherwise.

   (b) If $A(a)$ is quantifier-free, but does contain R, then, for example, $A(a) = R(a,a+1) \vee R(2a,4a)$, we have $[[A(n)]] = r_{n,n+1} \vee r_{2n,4n}$.

   (c) If $A(a) := \exists x \leq t(a)\ B(a,x)$, then $[[A(n)]] = \bigvee_{m\leq t(n)} [[B(m)]]$.

   (d) If $A(a) := \forall x \leq t(a)\ B(a,x)$, then $[[A(n)]] = \bigwedge_{m\leq t(n)} [[B(m)]]$.

3. Patch together an $\mathbf{AC}^0$-Frege proof from the translated lines (main step: unwind induction via repeated cut rules).