Bounded Arithmetic, Part III

Raheleh Jalali

Proof Society 2025

Outline

Witness

For Step 2, we need a notion of a witness. Witnesses are just values for the x that the formula $(\exists x \leqslant t)\phi(\bar{y}, x)$ is true, where ϕ is sharply bounded.

Witness

For Step 2, we need a notion of a witness. Witnesses are just values for the x that the formula $(\exists x \leqslant t)\phi(\bar{y}, x)$ is true, where ϕ is sharply bounded.

Definition

Let $A(\bar{c})$ be a formula of the above form. The predicate $\mathrm{Wit}_A(\bar{c}, u)$ is defined so that

- If A is $(\exists x \leqslant t)B(\bar{c}, x)$, $B \in \Delta_0^b$, then $\mathrm{Wit}_A(\bar{c}, u)$ is the formula

$$u \leqslant t \wedge B(\bar{c}, u).$$

Witness

For Step 2, we need a notion of a witness. Witnesses are just values for the x that the formula $(\exists x \leqslant t)\phi(\bar{y}, x)$ is true, where ϕ is sharply bounded.

Definition

Let $A(\bar{c})$ be a formula of the above form. The predicate $\mathrm{Wit}_A(\bar{c}, u)$ is defined so that

- If A is $(\exists x \leqslant t)B(\bar{c}, x)$, $B \in \Delta_0^b$, then $\mathrm{Wit}_A(\bar{c}, u)$ is the formula

$$u \leqslant t \wedge B(\bar{c}, u).$$

- If A is in Δ_0^b, then $\mathrm{Wit}_A(\bar{c}, u)$ is just $A(\bar{c})$.

Witness

For Step 2, we need a notion of a witness. Witnesses are just values for the x that the formula $(\exists x \leqslant t)\phi(\bar{y}, x)$ is true, where ϕ is sharply bounded.

Definition

Let $A(\bar{c})$ be a formula of the above form. The predicate $\mathrm{Wit}_A(\bar{c}, u)$ is defined so that

- If A is $(\exists x \leqslant t)B(\bar{c}, x)$, $B \in \Delta_0^b$, then $\mathrm{Wit}_A(\bar{c}, u)$ is the formula

$$u \leqslant t \wedge B(\bar{c}, u).$$

- If A is in Δ_0^b, then $\mathrm{Wit}_A(\bar{c}, u)$ is just $A(\bar{c})$.

We immediately have:

Lemma

- $A(\bar{c}) \leftrightarrow (\exists u) Wit_A(\bar{c}, u)$ *in* S_2^1.
- Wit_A *is a* Δ_0^b*-formula.*

Theorem (Witnessing Lemma)

Let $\Gamma \Rightarrow \Delta$ be an S_2^1-provable sequent of Σ_1^b formulas with free variables $\bar{c}$. Then there is a polynomial time function $f(\bar{c}, \bar{u})$ such that

$$\bigwedge_{\gamma_i \in \Gamma} Wit_{\gamma_i}(\bar{c}, u_i) \rightarrow \bigvee_{\delta_j \in \Delta} Wit_{\delta_j}(\bar{c}, f(\bar{c}, \bar{u})).$$

Proof sketch (of Witnessing Lemma (for S_2^1))

By induction on the number of lines in a free-cut free S_2^1-proof of $\Gamma \Rightarrow \Delta$, where all the formulas in Γ, Δ are Σ_1^b.

Proof sketch (of Witnessing Lemma (for S_2^1))

By induction on the number of lines in a free-cut free S_2^1-proof of $\Gamma \Rightarrow \Delta$, where all the formulas in Γ, Δ are Σ_1^b. Thus (simple case):

$$S_2^1 \vdash (\exists x_1 \leqslant t_1)\gamma_1(x_1, c), \ldots, (\exists x_k \leqslant t_k)\gamma_k(x_k, c) \Rightarrow$$

$$(\exists y_1 \leqslant s_1)\delta_1(y_1, c), \ldots, (\exists y_\ell \leqslant s_\ell)\delta_\ell(y_\ell, c)$$

Proof sketch (of Witnessing Lemma (for S_2^1))

By induction on the number of lines in a free-cut free S_2^1-proof of $\Gamma \Rightarrow \Delta$, where all the formulas in Γ, Δ are Σ_1^b. Thus (simple case):

$$S_2^1 \vdash (\exists x_1 \leqslant t_1)\gamma_1(x_1, c), \ldots, (\exists x_k \leqslant t_k)\gamma_k(x_k, c) \Rightarrow$$

$$(\exists y_1 \leqslant s_1)\delta_1(y_1, c), \ldots, (\exists y_\ell \leqslant s_\ell)\delta_\ell(y_\ell, c)$$

- Inner formulas, γ_i's, δ_j's, are sharply bounded; take polytime to check.

Proof sketch (of Witnessing Lemma (for S_2^1))

By induction on the number of lines in a free-cut free S_2^1-proof of $\Gamma \Rightarrow \Delta$, where all the formulas in Γ, Δ are Σ_1^b. Thus (simple case):

$$S_2^1 \vdash (\exists x_1 \leqslant t_1)\gamma_1(x_1, c), \ldots, (\exists x_k \leqslant t_k)\gamma_k(x_k, c) \Rightarrow$$

$$(\exists y_1 \leqslant s_1)\delta_1(y_1, c), \ldots, (\exists y_\ell \leqslant s_\ell)\delta_\ell(y_\ell, c)$$

- Inner formulas, γ_i's, δ_j's, are sharply bounded; take polytime to check.
- Witnessing lemma: there is a polynomial time procedure f such that
  - f takes $c, x_1, \ldots, x_k$, as input. These have the property that they make all γ_i's true, i.e., satisfy $\bigwedge_{i=1}^{k} \gamma_i(x_i, c)$.
  - f outputs y, which is a witness to one of δ_j's.

Proof sketch (of Witnessing Lemma (for S_2^1))

By induction on the number of lines in a free-cut free S_2^1-proof of $\Gamma \Rightarrow \Delta$, where all the formulas in Γ, Δ are Σ_1^b. Thus (simple case):

$$S_2^1 \vdash (\exists x_1 \leqslant t_1)\gamma_1(x_1, c), \ldots, (\exists x_k \leqslant t_k)\gamma_k(x_k, c) \Rightarrow$$

$$(\exists y_1 \leqslant s_1)\delta_1(y_1, c), \ldots, (\exists y_\ell \leqslant s_\ell)\delta_\ell(y_\ell, c)$$

- Inner formulas, γ_i's, δ_j's, are sharply bounded; take polytime to check.
- Witnessing lemma: there is a polynomial time procedure f such that
  - f takes $c, x_1, \ldots, x_k$, as input. These have the property that they make all γ_i's true, i.e., satisfy $\bigwedge_{i=1}^{k} \gamma_i(x_i, c)$.
  - f outputs y, which is a witness to one of δ_j's.

As an example, suppose that the proof ends with

$$\frac{B(\lfloor \frac{1}{2}a \rfloor, c) \Rightarrow B(a, c)}{B(0, c) \Rightarrow B(t, c)}$$

Proof.

Use IH, and limited recursion on notation to define f. By the induction hypothesis, there is a witness function g for the premise.

Proof.

Use IH, and limited recursion on notation to define f. By the induction hypothesis, there is a witness function g for the premise.
Now define f by recursion on notation as

$$f(w, 0, c) = w$$

$$f(w, a, c) = g(f(w, \lfloor \frac{1}{2} a \rfloor, c), a, c)$$

Proof.

Use IH, and limited recursion on notation to define f. By the induction hypothesis, there is a witness function g for the premise.
Now define f by recursion on notation as

$$f(w, 0, c) = w$$
$$f(w, a, c) = g(f(w, \lfloor \frac{1}{2} a \rfloor, c), a, c)$$

We show that f is polynomial time computable by showing that $|f|$ is bounded.

□

Proving Main Theorem using Witnessing Lemma

Theorem

Let f be Σ_1^b-defined by S_2^1. Then f is polytime computable.

Proving Main Theorem using Witnessing Lemma

Theorem

Let f be Σ^b_1-defined by S^1_2. Then f is polytime computable.

Importing Witnessing Lemma.

Recall by Step 1:

$$S^1_2 \vdash \Rightarrow (\exists y \leqslant t)\, A(c, y)$$

by a free-cut free π, where $A \in \Sigma^b_1$.

Proving Main Theorem using Witnessing Lemma

Theorem

Let f be Σ_1^b-defined by S_2^1. Then f is polytime computable.

Importing Witnessing Lemma.

Recall by Step 1:

$$S_2^1 \vdash \Rightarrow (\exists y \leqslant t)\ A(c, y)$$

by a free-cut free π, where $A \in \Sigma_1^b$.
Apply Witnessing Lemma to the last line of the proof
$(\Rightarrow (\exists y \leqslant t) A(y, c))$:

Proving Main Theorem using Witnessing Lemma

Theorem

Let f be Σ_1^b-defined by S_2^1. Then f is polytime computable.

Importing Witnessing Lemma.

Recall by Step 1:

$$S_2^1 \vdash \Rightarrow (\exists y \leqslant t)\, A(c, y)$$

by a free-cut free π, where $A \in \Sigma_1^b$.
Apply Witnessing Lemma to the last line of the proof
$(\Rightarrow (\exists y \leqslant t) A(y, c))$: Given c and witnesses for all the formulas on the left, we can find in polynomial time a witness for y on the right. So, we get a polytime function of c.

□

Outline

We have: $S_2^1 \subseteq T_2^1 \preceq_{\forall \mathbf{\Sigma}_2^b} S_2^2 \subseteq T_2^2 \preceq_{\forall \mathbf{\Sigma}_3^b} S_2^3 \subseteq \ldots$

We have: $S_2^1 \subseteq T_2^1 \preceq_{\forall\Sigma_2^b} S_2^2 \subseteq T_2^2 \preceq_{\forall\Sigma_3^b} S_2^3 \subseteq \ldots$

We already discussed that T_2^i contains S_2^i. Now:

$$T_2^i \preceq_{\forall\Sigma_2^b} S_2^{i+1}: \quad S_2^{i+1} \text{ is } \forall\Sigma_{i+1}^b \text{ conservative over } T_2^i$$

We have: $S_2^1 \subseteq T_2^1 \preceq_{\forall\Sigma_2^b} S_2^2 \subseteq T_2^2 \preceq_{\forall\Sigma_3^b} S_2^3 \subseteq \ldots$

We already discussed that T_2^i contains S_2^i. Now:

$$T_2^i \preceq_{\forall\Sigma_2^b} S_2^{i+1} : \quad S_2^{i+1} \text{ is } \forall\Sigma_{i+1}^b \text{ conservative over } T_2^i$$

Two parts of being $\forall\Sigma_{i+1}^b$ conservative are:

1. $T_2^i \subseteq S_2^{i+1}$, i.e., S_2^{i+1} is a possibly stronger, at least no weaker, theory containing T_2^i.
2. if ϕ is of the form $\forall \bar{x}\psi$ with $\psi \in \Sigma_{i+1}^b$ and if $S_2^{i+1} \vdash \phi$ then $T_2^i \vdash \phi$.

We have: $S_2^1 \subseteq T_2^1 \preceq_{\forall\Sigma_2^b} S_2^2 \subseteq T_2^2 \preceq_{\forall\Sigma_3^b} S_2^3 \subseteq \ldots$

We already discussed that T_2^i contains S_2^i. Now:

$$T_2^i \preceq_{\forall\Sigma_2^b} S_2^{i+1}: \quad S_2^{i+1} \text{ is } \forall\Sigma_{i+1}^b \text{ conservative over } T_2^i$$

Two parts of being $\forall\Sigma_{i+1}^b$ conservative are:

1. $T_2^i \subseteq S_2^{i+1}$, i.e., S_2^{i+1} is a possibly stronger, at least no weaker, theory containing T_2^i.
2. if ϕ is of the form $\forall \bar{x}\psi$ with $\psi \in \Sigma_{i+1}^b$ and if $S_2^{i+1} \vdash \phi$ then $T_2^i \vdash \phi$.

$\preceq_{\forall\Sigma_{i+1}^b}$ means: subset plus nothing new of this complexity class (i.e., theories are equal on the class $\forall\Sigma_{i+1}^b$). The stronger theory adds no new low-complexity facts.

Theorem (Buss '90)

Let $i \geqslant 1$.

1. S_2^{i+1} *is* $\forall\Sigma^b_{i+1}$ *conservative over* T_2^i. $(T_2^i \preceq_{\forall\Sigma^b_2} S_2^{i+1})$
2. *In particular,* T_2^i *can* Σ^b_{i+1} *define precisely the functions in* $FP^{\Sigma_i^p}$.

Theorem (Buss '90)

Let $i \geqslant 1$.

1. *S_2^{i+1} is $\forall\Sigma_{i+1}^b$ conservative over T_2^i. ($T_2^i \preceq_{\forall\Sigma_2^b} S_2^{i+1}$)*
2. *In particular, T_2^i can Σ_{i+1}^b define precisely the functions in $FP^{\Sigma_i^p}$.*

Proof idea

With some work, one can show that the witnessing lemma carries over to the base theory when using T_2^i in place of S_2^{i+1}.

This highlights once more the strength of the witnessing lemma: it precisely characterizes what can be done in these theories.

If T_2^i proves the collapse of the polynomial hierarchy, then the Buss hierarchy of theories collapse.

If T_2^i proves the collapse of the polynomial hierarchy, then the Buss hierarchy of theories collapse. The reason simply is that if T_2^i believes that $\{\Sigma_k^b\}_{k=j}^{\infty}$ collapses to Σ_j^b, then $T_2^{\max\{i,j\}}$ proves all bounded inductions.

If T_2^i proves the collapse of the polynomial hierarchy, then the Buss hierarchy of theories collapse. The reason simply is that if T_2^i believes that $\{\Sigma_k^b\}_{k=j}^{\infty}$ collapses to Σ_j^b, then $T_2^{\max\{i,j\}}$ proves all bounded inductions. Conversely:

Theorem (Krajicek-Pudlak-Takeuti'91, Buss'95, Zambella'96)

If $T_2^i = S_2^{i+1}$, then the polynomial hierarchy collapses with $\Sigma_{i+2}^p = \Pi_{i+2}^p$. Moreover, T_2^i can prove that the polynomial time hierarchy collapses (provably) to $B(\Sigma_{i+2}^b)$.

If T_2^i proves the collapse of the polynomial hierarchy, then the Buss hierarchy of theories collapse. The reason simply is that if T_2^i believes that $\{\Sigma_k^b\}_{k=j}^{\infty}$ collapses to Σ_j^b, then $T_2^{\max\{i,j\}}$ proves all bounded inductions. Conversely:

Theorem (Krajicek-Pudlak-Takeuti'91, Buss'95, Zambella'96)

If $T_2^i = S_2^{i+1}$, then the polynomial hierarchy collapses with $\Sigma_{i+2}^p = \Pi_{i+2}^p$. Moreover, T_2^i can prove that the polynomial time hierarchy collapses (provably) to $B(\Sigma_{i+2}^b)$.

- $B(\Sigma_{i+2}^b)$ means the hierarchy collapses to to uniform boolean combinations of Σ_{i+2}^b and thus to $\Sigma_{i+3}^b = \Pi_{i+3}^b$.
- In such case, the hierarchy collapses precisely to these levels.
- Conjecture: the hierarchy is strict.
- But, separating these theories means that they do not prove the collapse of the polynomial hierarchy. A major breakthrough.

Outline

Relation to propositional proof complexity

- Question in computational complexity: What can be computed or expressed?

Relation to propositional proof complexity

- Question in computational complexity: What can be computed or expressed?
- Question in BA (and in general in proof complexity): add to the above question that how hard it is to prove that the expression/computation is correct or has some specific properties.

Relation to propositional proof complexity

- Question in computational complexity: What can be computed or expressed?
- Question in BA (and in general in proof complexity): add to the above question that how hard it is to prove that the expression/computation is correct or has some specific properties.
- Theories of BA are uniform counterparts to standard propositional proof systems (like Hilbert-style) and are in particular useful for constructing short (i.e., polynomial size) proofs in these systems.

Relation to propositional proof complexity

- Question in computational complexity: What can be computed or expressed?
- Question in BA (and in general in proof complexity): add to the above question that how hard it is to prove that the expression/computation is correct or has some specific properties.
- Theories of BA are uniform counterparts to standard propositional proof systems (like Hilbert-style) and are in particular useful for constructing short (i.e., polynomial size) proofs in these systems.
- As Turing machines are uniform versions of non-uniform models of computation (like circuits), theories of bounded arithmetic are uniform versions of propositional proof systems.

Propositional proof systems

There's two different kinds of connections to propositional proof systems.

- Translations from bounded arithmetic to propositional logic
  - "Cook-style" translations.
  - "Paris-Wilkie style" translations.

Abstract proof systems

There are many different ways to formalize the concept of a proof. The basic property: being a proof must be easy to check (polynomial time).

Abstract proof systems

There are many different ways to formalize the concept of a proof. The basic property: being a proof must be easy to check (polynomial time).

Definition (Cook-Reckhow '79)

A *propositional proof system* is any polynomial time computable function

$$f : \{0,1\}^* \to \{0,1\}^*$$

such that

$$TAUT = Rng(f).$$

Any $w \in \{0,1\}^*$ such that $f(w) = A$ is called an f-proof of A.

Abstract proof systems

There are many different ways to formalize the concept of a proof. The basic property: being a proof must be easy to check (polynomial time).

Definition (Cook-Reckhow '79)

A *propositional proof system* is any polynomial time computable function

$$f : \{0,1\}^* \to \{0,1\}^*$$

such that

$$TAUT = Rng(f).$$

Any $w \in \{0,1\}^*$ such that $f(w) = A$ is called an f-proof of A.

- f is sound: its range contains *only* tautologies, $Rng(f) \subseteq \mathrm{TAUT}$.

Abstract proof systems

There are many different ways to formalize the concept of a proof. The basic property: being a proof must be easy to check (polynomial time).

Definition (Cook-Reckhow '79)

A *propositional proof system* is any polynomial time computable function

$$f : \{0,1\}^* \to \{0,1\}^*$$

such that

$$TAUT = Rng(f).$$

Any $w \in \{0,1\}^*$ such that $f(w) = A$ is called an f-proof of A.

- f is sound: its range contains *only* tautologies, $Rng(f) \subseteq \mathrm{TAUT}$.
- f is complete: its range contains *all* tautologies, $\mathrm{TAUT} \subseteq Rng(f)$.

Abstract proof systems

There are many different ways to formalize the concept of a proof. The basic property: being a proof must be easy to check (polynomial time).

Definition (Cook-Reckhow '79)

A *propositional proof system* is any polynomial time computable function

$$f : \{0,1\}^* \to \{0,1\}^*$$

such that

$$TAUT = Rng(f).$$

Any $w \in \{0,1\}^*$ such that $f(w) = A$ is called an f-proof of A.

- f is sound: its range contains *only* tautologies, $Rng(f) \subseteq \mathrm{TAUT}$.
- f is complete: its range contains *all* tautologies, $\mathrm{TAUT} \subseteq Rng(f)$.

This definition sounds like a fairly non-intuitive way to define a proof system but it's also a very general way.

Frege system (F)

Some examples of propositional proof systems.
Frege systems are the usual "textbook" proof systems for propositional logic, using modus ponens as their only rule of inference.
Connectives: $\wedge$, $\vee$, $\neg$, and $\rightarrow$.
Modus ponens (MP):

$$\frac{A \quad A \rightarrow B}{B}$$

Axioms: Finite set of axiom schemes, e.g.: $A \wedge B \rightarrow A$ or $A \rightarrow (B \rightarrow A)$

Frege system (F)

Some examples of propositional proof systems.
Frege systems are the usual "textbook" proof systems for propositional logic, using modus ponens as their only rule of inference.
Connectives: $\wedge$, $\vee$, $\neg$, and $\rightarrow$.
Modus ponens (MP):

$$\frac{A \quad A \rightarrow B}{B}$$

Axioms: Finite set of axiom schemes, e.g.: $A \wedge B \rightarrow A$ or $A \rightarrow (B \rightarrow A)$
Frege proofs: A proof of B from a set of assumptions X, denoted by $X \vdash_F B$, is $B_1, \ldots, B_m = B$ such that each $B_i \in X$, or is inferred from some B_j, $j < i$, by a substitution instance of MP. The formulas B_i are called *lines* of the proof.

Frege system (F)

Some examples of propositional proof systems.
Frege systems are the usual "textbook" proof systems for propositional logic, using modus ponens as their only rule of inference.
Connectives: $\wedge$, $\vee$, $\neg$, and $\rightarrow$.
Modus ponens (MP):

$$\frac{A \quad A \rightarrow B}{B}$$

Axioms: Finite set of axiom schemes, e.g.: $A \wedge B \rightarrow A$ or $A \rightarrow (B \rightarrow A)$
Frege proofs: A proof of B from a set of assumptions X, denoted by $X \vdash_F B$, is $B_1, \ldots, B_m = B$ such that each $B_i \in X$, or is inferred from some B_j, $j < i$, by a substitution instance of MP. The formulas B_i are called *lines* of the proof.

We're interested in the complexity of the proofs, i.e., the *size* of the proofs, which is the number of symbols in the proof.

Extended Frege system (EF)

An extended Frege system is a Frege system augmented with the extension axiom, which formalizes the intuitive practice of naming longer formulas so that they can be referred to succinctly in the proof.

Extended Frege system (EF)

An extended Frege system is a Frege system augmented with the extension axiom, which formalizes the intuitive practice of naming longer formulas so that they can be referred to succinctly in the proof.

- Pick a new variable p and a formula A that does not contain p, and let p be an abbreviation for A, i.e., add $p \leftrightarrow A$ as a new axiom.

Extended Frege system (EF)

An extended Frege system is a Frege system augmented with the extension axiom, which formalizes the intuitive practice of naming longer formulas so that they can be referred to succinctly in the proof.

- Pick a new variable p and a formula A that does not contain p, and let p be an abbreviation for A, i.e., add $p \leftrightarrow A$ as a new axiom.
- In an extended Frege proof, p is a new variable, i.e., does not appear in any lines before $p \leftrightarrow A$, nor in A, nor in the last line of the proof.

Extended Frege system (EF)

An extended Frege system is a Frege system augmented with the extension axiom, which formalizes the intuitive practice of naming longer formulas so that they can be referred to succinctly in the proof.

- Pick a new variable p and a formula A that does not contain p, and let p be an abbreviation for A, i.e., add $p \leftrightarrow A$ as a new axiom.
- In an extended Frege proof, p is a new variable, i.e., does not appear in any lines before $p \leftrightarrow A$, nor in A, nor in the last line of the proof.

There are two reasons why there could be a lot of symbols in a proof:

- There are many steps/lines in a proof.
- Formulas in the proof are very large.

Extended Frege system (EF)

An extended Frege system is a Frege system augmented with the extension axiom, which formalizes the intuitive practice of naming longer formulas so that they can be referred to succinctly in the proof.

- Pick a new variable p and a formula A that does not contain p, and let p be an abbreviation for A, i.e., add $p \leftrightarrow A$ as a new axiom.
- In an extended Frege proof, p is a new variable, i.e., does not appear in any lines before $p \leftrightarrow A$, nor in A, nor in the last line of the proof.

There are two reasons why there could be a lot of symbols in a proof:

- There are many steps/lines in a proof.
- Formulas in the proof are very large.

Extension is a first attempt to reduce size of the formulas; replace large formulas with new variables, thus not repeating the same formula. Another point: You can iterate extension, to get exponentially big formulas represented by a single variable.

Sequent calculus

The sequent calculus will be particularly nice for dealing with translations into the constant depth proof setting. We work with propositional LK, i.e., LK minus the quantifier inference rules.

Observation: the sequent calculus is equivalent to Frege systems. One can convert a Frege proof into a sequent calculus proof, and vice versa, with a polynomial blow up in proof size.

Cook's translation for *PV*

- Cook introduced the equational theory PV (Polynomially Verifiable), which has a function symbol for every polynomial time function.

Cook's translation for *PV*

- Cook introduced the equational theory PV (Polynomially Verifiable), which has a function symbol for every polynomial time function.
- Cook used a translation and characterized the logical strength of PV in terms of provability in EF.

Cook's translation for *PV*

- Cook introduced the equational theory PV (Polynomially Verifiable), which has a function symbol for every polynomial time function.
- Cook used a translation and characterized the logical strength of PV in terms of provability in EF.
- It turns out that PV has a logical strength extremely close to S_2^1. Cook's result is earlier than the development of S_2^1.

Cook's translation for *PV*

- Cook introduced the equational theory PV (Polynomially Verifiable), which has a function symbol for every polynomial time function.
- Cook used a translation and characterized the logical strength of PV in terms of provability in EF.
- It turns out that PV has a logical strength extremely close to S_2^1. Cook's result is earlier than the development of S_2^1.

For a p-time identity $f(x) = g(x)$, where f, g are p-time functions, define a family of propositional formulas $[\![f = g]\!]_n$, where n is an integer.

Cook's translation for *PV*

- Cook introduced the equational theory PV (Polynomially Verifiable), which has a function symbol for every polynomial time function.
- Cook used a translation and characterized the logical strength of PV in terms of provability in EF.
- It turns out that PV has a logical strength extremely close to S_2^1. Cook's result is earlier than the development of S_2^1.

For a p-time identity $f(x) = g(x)$, where f, g are p-time functions, define a family of propositional formulas $[\![f = g]\!]_n$, where n is an integer.

- $[\![f = g]\!]_n$ expresses that $f(x) = g(x)$ for any $|x| \leqslant n$.

Cook's translation for *PV*

- Cook introduced the equational theory PV (Polynomially Verifiable), which has a function symbol for every polynomial time function.
- Cook used a translation and characterized the logical strength of PV in terms of provability in EF.
- It turns out that PV has a logical strength extremely close to S_2^1. Cook's result is earlier than the development of S_2^1.

For a p-time identity $f(x) = g(x)$, where f, g are p-time functions, define a family of propositional formulas $[\![f = g]\!]_n$, where n is an integer.

- $[\![f = g]\!]_n$ expresses that $f(x) = g(x)$ for any $|x| \leqslant n$.
- The variables in $[\![f = g]\!]_n$ are the bits $x_0, \ldots, x_{n-1}$ of x.

Cook's translation for *PV*

- Cook introduced the equational theory PV (Polynomially Verifiable), which has a function symbol for every polynomial time function.
- Cook used a translation and characterized the logical strength of PV in terms of provability in EF.
- It turns out that PV has a logical strength extremely close to S^1_2. Cook's result is earlier than the development of S^1_2.

For a p-time identity $f(x) = g(x)$, where f, g are p-time functions, define a family of propositional formulas $[\![f = g]\!]_n$, where n is an integer.

- $[\![f = g]\!]_n$ expresses that $f(x) = g(x)$ for any $|x| \leqslant n$.
- The variables in $[\![f = g]\!]_n$ are the bits $x_0, \ldots, x_{n-1}$ of x.

Theorem (Cook '75)

If $PV \vdash f(x) = g(x)$, then the formulas $[\![f = g]\!]_n$ have polynomial size extended Frege proofs.

Cook translation for S_2^1

The results lift to S_2^1, however, the construction changes a bit:

- Suppose $S_2^1 \vdash \forall x A(x)$, where $A(x) \in \Sigma_0^b$.
- For $n > 0$, form polynomial size propositional formulas $[\![A]\!]_n$.
- $[\![A]\!]_n$ has Boolean variables $x_0, \ldots, x_{n-1}$ representing the bits of x, where $|x| \leqslant n$.
- $[\![A]\!]_n$ expresses that "$A(x)$ is true" whenever $|x| \leqslant n$.
- $[\![A]\!]_n$ have polynomial size EF proofs.

Rather than formally define $[\![A]\!]_n$, we give an example.

Example: $[\![A(x)]\!]_n$: $[\![(\forall a \leqslant |x|)(a - 1 < x)]\!]_n$

Let $x = x_{n-1} \dots x_0$ and $a = a_{n-1} \dots a_0$ be two n-bit integers.

$$[\![x = a]\!]_n := \bigwedge_{i=0}^{n-1} (x_i \leftrightarrow a_i)$$

$$[\![x < a]\!]_n := \bigvee_{i=0}^{n-1} \left((a_i \wedge \neg x_i) \wedge \bigwedge_{j=i+1}^{n-1} (x_j \leftrightarrow a_j) \right)$$

$$[\![x \leqslant a]\!]_n := [\![x < a]\!]_n \vee [\![x = a]\!]_n$$

- i-th bit of $x - 1$: $(x-1)_i \leftrightarrow (x_i \leftrightarrow \bigvee_{j=0}^{i-1} x_j) \wedge [\![x \neq 0]\!]_n$
- i-th bit of $|x|$: $\bigvee_{j \leqslant n, (j)_i = 1} (x_j \wedge \bigvee_{k=j+1}^{n} \neg x_k)$

$$[\![(\forall a \leqslant |x|)(a - 1 < x)]\!]_n := \bigwedge_{a=0}^{n} ([\![a \leqslant |x|]\!]_n \rightarrow [\![a - 1 < x]\!]_n).$$

The sharply bounded quantifier $(\forall a \leqslant |x|)$ becomes a conjunction.

Comments

- n is a length of a number and is fixed and $|x| \leqslant n$.

Comments

- n is a length of a number and is fixed and $|x| \leqslant n$.
- We have a sharply bounded quantifier; a ranges over values up to $|x|$.

Comments

- n is a length of a number and is fixed and $|x| \leqslant n$.
- We have a sharply bounded quantifier; a ranges over values up to $|x|$.
- Convert $\forall a \leqslant |x|$ to a conjunction over all possible values of a: There are only polynomially many of them.

Comments

- n is a length of a number and is fixed and $|x| \leqslant n$.
- We have a sharply bounded quantifier; a ranges over values up to $|x|$.
- Convert $\forall a \leqslant |x|$ to a conjunction over all possible values of a: There are only polynomially many of them.
- For each a, form the formula corresponding to the value of a by hard wiring its bits, starting from $a = 0$, then $a = 1$, so on.

Comments

- n is a length of a number and is fixed and $|x| \leqslant n$.
- We have a sharply bounded quantifier; a ranges over values up to $|x|$.
- Convert $\forall a \leqslant |x|$ to a conjunction over all possible values of a: There are only polynomially many of them.
- For each a, form the formula corresponding to the value of a by hard wiring its bits, starting from $a = 0$, then $a = 1$, so on.
- Idea: $\forall, \exists$ are all sharply bounded. Converting them into $\wedge, \vee$, we only need to consider a small amount of values (i.e., polynomially many). Hence, we get the translation and its size will be polynomial.

Comments

- n is a length of a number and is fixed and $|x| \leqslant n$.
- We have a sharply bounded quantifier; a ranges over values up to $|x|$.
- Convert $\forall a \leqslant |x|$ to a conjunction over all possible values of a: There are only polynomially many of them.
- For each a, form the formula corresponding to the value of a by hard wiring its bits, starting from $a = 0$, then $a = 1$, so on.
- Idea: $\forall, \exists$ are all sharply bounded. Converting them into $\wedge, \vee$, we only need to consider a small amount of values (i.e., polynomially many). Hence, we get the translation and its size will be polynomial.
- We look at: not the first n numbers, the numbers with n bits.

- Make sure the used function symbols are defined in the language. There is no “minus one” in the language, but can be easily expressed using polynomial size formulas. The most difficult case: multiplication. It can be handled using carry-save addition.

- Make sure the used function symbols are defined in the language. There is no "minus one" in the language, but can be easily expressed using polynomial size formulas. The most difficult case: multiplication. It can be handled using carry-save addition.
- We form these formulas independently of the theorem; the theorem only states that the resulting formulas have polynomial-size extended Frege proofs, which in turn come from the S_2^1 proof.

Theorem (essentially (Cook'75))

If $S_2^1 \vdash (\forall x)A(x)$, where $A(x)$ is in Δ_0^b (or a polynomial time identity), then the tautologies $[\![A(x)]\!]_n$ have polynomial size extended Frege proofs.

Theorem (essentially (Cook'75))

If $S_2^1 \vdash (\forall x)A(x)$, where $A(x)$ is in Δ_0^b (or a polynomial time identity), then the tautologies $[\![A(x)]\!]_n$ have polynomial size extended Frege proofs.

Proof sketch.

Witnessing Lemma again.

- Take a free cut-free proof. All the formulas are Δ_0^b. Do extra work.
- You actually extract from the S_2^1 proof, the polynomial size EF proof.

Theorem (essentially (Cook'75))

If $S_2^1 \vdash (\forall x)A(x)$, where $A(x)$ is in Δ_0^b (or a polynomial time identity), then the tautologies $[\![A(x)]\!]_n$ have polynomial size extended Frege proofs.

Proof sketch.

Witnessing Lemma again.

- Take a free cut-free proof. All the formulas are Δ_0^b. Do extra work.
- You actually extract from the S_2^1 proof, the polynomial size EF proof.
- The intuition is: by witnessing theorem, there are polynomial time functions, that witness these sequents.

Theorem (essentially (Cook'75))

If $S_2^1 \vdash (\forall x)A(x)$, where $A(x)$ is in Δ_0^b (or a polynomial time identity), then the tautologies $[\![A(x)]\!]_n$ have polynomial size extended Frege proofs.

Proof sketch.

Witnessing Lemma again.

- Take a free cut-free proof. All the formulas are Δ_0^b. Do extra work.
- You actually extract from the S_2^1 proof, the polynomial size EF proof.
- The intuition is: by witnessing theorem, there are polynomial time functions, that witness these sequents.
- polynomial time functions have polynomial size circuits.

Theorem (essentially (Cook'75))

If $S_2^1 \vdash (\forall x)A(x)$, where $A(x)$ is in Δ_0^b (or a polynomial time identity), then the tautologies $[\![A(x)]\!]_n$ have polynomial size extended Frege proofs.

Proof sketch.

Witnessing Lemma again.

- Take a free cut-free proof. All the formulas are Δ_0^b. Do extra work.
- You actually extract from the S_2^1 proof, the polynomial size EF proof.
- The intuition is: by witnessing theorem, there are polynomial time functions, that witness these sequents.
- polynomial time functions have polynomial size circuits.
- polynomial size circuits have values that can be defined using extension polynomially many times.
- The arguments in S_2^1 can be transformed into arguments in EF setting.

□

Cook proved more: We can prove the consistency of EF proofs in S_2^1.

Theorem (Cook'75)

- $S_2^1 \vdash Con(EF)$ *(the consistency of EF).*
- *For any propositional proof system* G*, if* $S_2^1 \vdash Con(G)$*, then EF p-simulates* G*.*

Cook proved more: We can prove the consistency of EF proofs in S_2^1.

Theorem (Cook'75)

- $S_2^1 \vdash Con(EF)$ *(the consistency of EF).*
- *For any propositional proof system G, if $S_2^1 \vdash Con(G)$, then EF p-simulates G.*

Recall: p-simulate means for a proof π of a formula A in G, we can find a proof for A in EF whose size is polynomial in $|\pi|$. This conversion only makes a polynomial blow up.

Cook proved more: We can prove the consistency of EF proofs in S_2^1.

Theorem (Cook'75)

- $S_2^1 \vdash Con(EF)$ *(the consistency of EF).*
- *For any propositional proof system G, if $S_2^1 \vdash Con(G)$, then EF p-simulates G.*

Recall: p-simulate means for a proof π of a formula A in G, we can find a proof for A in EF whose size is polynomial in $|\pi|$. This conversion only makes a polynomial blow up.

Theorem: EF is the strongest pps whose consistency is provable by S_2^1.

We are thinking about provability in a system vs provability in another. Similar to the computability in uniform settings (Turing Machines) vs non-uniform settings (circuits), EF proofs are non-uniform proofs of S_2^1.

Outline

Second-order objects

- Second order language: First order variables range over non-negative integers. Second order variables $\alpha, \beta, \ldots$ range over sets of integers.
- From a computational point of view, adding α to the language is like adding an oracle symbol and you can query the oracle.
- We talked about adding a new polynomial time function symbol to the language and being conservative.
- Here, we add a new symbol α without its defining axioms; we only allow it to appear in induction formulas.

Example

Example

$$S_2^1(\alpha) \vdash \forall x \exists y(\alpha(0) \wedge \neg\alpha(x) \rightarrow \alpha(y) \wedge \neg\alpha(y+1)).$$

Think of x as ranging over integers. We're saying that $\alpha(0)$ is true, $\alpha(x)$ is false implies there exists a y, where $\alpha(y)$ is true and $\alpha(y+1)$ is false:

0	y	$y+1$	x
$\alpha(0)$	$\alpha(y)$	$\neg\alpha(y+1)$	$\neg\alpha(x)$

Binary search

- This may be surprising at first: potentially searching from 0 to x is an exponentially long process.
- Binary search.
- Using a binary search algorithm you can ask whether $\alpha(\frac{x}{2})$; if it's true, go right, otherwise, go left. Then ask $\alpha(\frac{x}{4})$ and so forth till you find a y where $\alpha(y), \neg\alpha(y+1)$.
- A binary search algorithm, as it runs, queries whether α holds for particular values. It works for any α, always succeeds, and is p-time.

Getting ready for relativized S_2^i and T_2^i

- For relativized S_2^i and T_2^i: we do not take second order quantifiers; just consider first order logic augmented with second order variables.
- We treat second order predicates as being polynomial time computable; we allow them to be used freely in computations.
- p^α: polynomial time computable functions relative to an oracle for α. This is a polynomial time Turing machine which is allowed to query α as the oracle: at any point it can stop and write an integer on its quary tape and ask whether α holds for it.

Relativized versions of S_2^i and T_2^i

Definition ($\Sigma_i^b(\alpha)$ and $\Pi_i^b(\alpha)$)

$\Sigma_i^b(\alpha)$ and $\Pi_i^b(\alpha)$ are defined exactly like Σ_i^b and Π_i^b but now allowing atomic formulas $\alpha(t)$, i.e., quantifier free, which doesn't count towards the quantifier complexity.

Definition

- $S_2^i(\alpha)$ is: $BASIC + \Sigma_i^b(\alpha) - PIND$.
- $T_2^i(\alpha)$ is: $BASIC + \Sigma_i^b(\alpha) - IND$.
- $S_2(\alpha) = T_2(\alpha) = \cup_i T_2^i(\alpha)$.

Theorem

The $\Sigma_1^b(\alpha)$-definable functions of $S_2^1(\alpha)$ are precisely the functions in P^α (so α is an oracle).

Outline

Bounded depth Frege systems

There are three principal complexity notions for proofs:

(a) The number of *lines* or *steps*, $\lambda(\pi)$, in a proof $\pi = (A_1, \ldots, A_m)$ is m.

(b) The *length* or *size* of the proof is $n = |\pi| = \sum_{i=1}^{m} |A_i|$.

(c) The *depth* d of the proof is the maximum AND/OR depth of a formula A_i occurring in the proof.

The AND/OR depth of a formula A: Write A over the basis $\{\wedge, \vee, \neg\}$, and use DeMorgan's Laws to push the $\neg$'s on the variables only. Then count the maximum number of alternations between $\wedge$ and $\vee$ in a path from the top operand to a literal in the formula.

Definition (Bounded depth Frege systems)

The *depth-d* Frege system F_d is the fragment of a Frege proof system in which all lines are required to have logical depth $\leqslant d$.

The Paris-Wilkie translation

- Paris-Wilkie translation: first in 1985 and in the setting $I\Delta_0$.
- It transforms a $T_2^i(\alpha)$ proof, a proof at one of the levels of the hierarchy to *constant depth propositional proofs* of the sequent calculus LK.
- We work in systems with a second order variable α. Propositional variables x_j are no longer integers, but values for $\alpha(j)$.
- Bounded quantifiers convert to $\wedge$ and $\vee$.
- The depth of the propositional formula is approximately i.

Basic idea: A constant depth proof is equivalent to a constant alternation of $\wedge, \vee$. Take a $T_2^i(\alpha)$. It consists of formulas with i alternations of bounded first order quantifiers. Convert these quantifiers to $\wedge, \vee$, which will become the approximate depth i formulas.

Example: PW translation of PHP

- Pigeonhole principle: an important statement in complexity theory.
- Weak pigeonhole principle: there is no injective function f mapping $a+1$ objects into a objects.
- α will code the graph of f: $\alpha(x,y)$ means $f(x)=y$.

$\mathrm{PHP}^{\alpha}(a)$ stating $\alpha(x,y)$ does not violate the pigeonhole principle, is:

$$(\forall x \leqslant a)(\exists y < a)(\alpha(x,y)) \rightarrow (\exists x \leqslant a)(\exists x' < x)(\exists y < a)[\alpha(x,y) \wedge \alpha(x',y)].$$

For a fixed integer $n \in \mathbb{N}$, $[\![PHP^{\alpha}(a)]\!]_n^{PW}$ is the propositional formula:

$$\bigwedge_{i=0}^{n} \bigvee_{j=0}^{n-1} z_{i,j} \rightarrow \bigvee_{i=0}^{n} \bigvee_{i'=0}^{i-1} \bigvee_{j=0}^{n-1} (z_{i,j} \wedge z_{i',j}).$$

General principles for the translation:

- First order values are set to constants (like setting a to the constant number n), and evaluated to a fixed value. There are no Boolean variables for bits of first-order objects.
- The Boolean values $\alpha(\cdots)$ become propositional variables.

With a direct translation of the $T_2^i(\alpha)$ proof we obtain LK proofs:

Theorem (Paris-Wilkie translation)

Let $i \geqslant 2$ and $T_2^i(\alpha) \vdash \forall a B(a, \alpha)$, where $B(a, \alpha)$ is a $\Sigma^b_{i-2}(\alpha)$ formula. Then, there are quasipolynomial size LK proofs P_n of the propositional translations $[\![B(a, \alpha)]\!]_n^{PW}$.