A Propositional Proof System for Log Space

Steven Perron

Department of Computer Science, University of Toronto
sperron@cs.toronto.edu

Abstract. The proof system G_0^* of the quantified propositional calculus corresponds to NC^1, and G_1^* corresponds to P, but no formula-based proof system that corresponds log space reasoning has ever been developed. This paper does this by developing GL^*.

We begin by defining a class $\Sigma CNF(2)$ of quantified formulas that can be evaluated in log space. Then GL^* is defined as G_1^* with cuts restricted to $\Sigma CNF(2)$ formulas and no cut formula that is not quantifier free contains a non-parameter free variable.

To show that GL^* is strong enough to capture log space reasoning, we translate theorems of Σ_0^B-rec into a family of tautologies that have polynomial size GL^* proofs. Σ_0^B-rec is a theory of bounded arithmetic that is known to correspond to log space. To do the translation, we find an appropriate axiomatization of Σ_0^B-rec, and put Σ_0^B-rec proofs into a new normal form.

To show that GL^* is not too strong, we prove the soundness of GL^* in such a way that it can be formalized in Σ_0^B-rec. This is done by giving a log space algorithm that witnesses GL^* proofs.

1 Introduction

Recently there has been lots of research looking into the connection between computational complexity, bounded arithmetic, and propositional proof complexity. A recent survey on this topic can be found at [1]. In this paper, we give a method of restricting the proof system G^* to get a proof system GL^*, which corresponds to log space. The proof system G^* is a tree-like proof system for the quantified propositional calculus based on Gentzen's LK [2]. This affirms the belief of Cook that there exists a formula-based proof system that corresponds to log space [1]. Before this the only proof system for log space was based on liar games [3], and it has never been well developed.

The definition of GL^* is similar to the definition of G_i^*: it is obtain from G^* by restricting cuts. We restrict cuts to the class of formulas $\Sigma CNF(2)$. This class of formulas is closely related to $CNF(2)$ formulas. These are CNF formulas where no variable appears more than twice.

Another attempt to capture reasoning between NC^1 and P resorted to putting a bound on the depth of the proof and the number of cuts along a branch [4], but it is not obvious how to capture other complexity classes using this type of restriction. In contrast to that, it seems plausible that a proof system for NL could be defined in the same spirit as GL^*.

The rest of the paper is organized as follows. In the next section, we define the $\Sigma CNF(2)$ formulas and the proof system GL^*, and we give the reason for the restrictions on the cut formulas. Section 3 is devoted to translating theorems of Σ_0^B-rec into tautologies with polynomial size GL^* proofs. Σ_0^B-rec is a theory of bounded arithmetic introduced by Zambella [12] that is known to correspond to log space [1]. This proves GL^* is strong enough to capture log space reasoning. In section 4, we prove in Σ_0^B-rec that GL^* is sound. This is sometimes called the reflection principle. This tells us that GL^* does not capture reasoning for a higher complexity class.

This paper is based on my Masters thesis [9], where more details can be found.

2 The Proof System

The proof system PK is the Gentzen-style sequent calculus for propositional logic [5, 6]. The initial sequents are $\bot \rightarrow$, $\rightarrow \top$, and $A \rightarrow A$, for any propositional formula A. The rules of inference include structural rules, which are weakening, contraction, and exchange; propositional rules, which are used to add propositional connective to formulas in the sequents; and the cut rule, which infers $\Gamma \rightarrow \Delta$ from $A, \Gamma \rightarrow \Delta$ and $\Gamma \rightarrow \Delta, A$. We can then extend PK to the proof system G by adding rules for quantifiers over propositional variables [7]. Anytime you deal with quantifiers, you will have problems with the substituting terms or, in this case, formulas for variables. To help avoid these problems, we use the following convention.

Notation 1 *In propositional formulas, all bound variables will be z, $z_1, z_2, \ldots$ and will be called z-variables. The free variables will be x, $x_1, x_2 \ldots$ and will be called x-variables.*

This way we know, x-variables are never quantified. The quantifier rules are

$$\exists\text{-left}\ \frac{A(x), \Gamma \rightarrow \Delta}{\exists z A(z), \Gamma \rightarrow \Delta} \qquad \exists\text{-right}\ \frac{\Gamma \rightarrow \Delta, A(B)}{\Gamma \rightarrow \Delta, \exists z A(z)}$$

$$\forall\text{-left}\ \frac{\Gamma \rightarrow \Delta, A(x)}{\Gamma \rightarrow \Delta, \forall z A(z)} \qquad \forall\text{-right}\ \frac{A(B), \Gamma \rightarrow \Delta}{\forall z A(z), \Gamma \rightarrow \Delta}$$

where x does not appear in the bottom sequent of the $\exists$-left and $\forall$-right rules, and B is a Σ_0^q formula that does not mention any z-variable. The Σ_0^q formulas are propositional formulas that do not contain any quantifiers. In general, the Σ_i^q formulas are quantified propositional formulas in prenex form with at most $i-1$ quantifier alternations, beginning with $\exists$ on the outside. In these rules, x is called the *eigenvariable* and B is called the *target formula*. The proof system

G^* is G restricted to tree-like proofs. The fragment G_i is G with cuts restricted to Σ_i^q formulas, and G_i^* is G_i restricted to tree-like proofs [7].

These proof systems have been extensively studied. In [2], Krajicek and Pudlak showed a connection between G_i and Buss's theories T_2^i for $i > 0$. This result was shown for a different definition of G_i, but the result still holds [7]. Later a connection between G_i^* and S_2^i [8] was found. With the connection between these theories and the polynomial-time hierarchy, we get an indirect connection between the polynomial-time hierarchy and the proof systems. In particular, G_1^* is connected to P. Later the proof system G_0^* was shown to be directly connected to NC^1, and connections to the theory VNC^1 were also considered. Since our proof system is going to correspond to log space, it must be between G_0^* and G_1^*; the cut formulas must be some subset of Σ_1^q formulas. This subset is the $\Sigma CNF(2)$ formulas.

Definition 1 *The set of formulas $\Sigma CNF(2)$ is the smallest set*

1. *containing Σ_0^q,*
2. *containing every formula $\exists \boldsymbol{z}, \phi(\boldsymbol{z}, \boldsymbol{x})$ where (1) ϕ is a quantifier-free CNF formula $\bigwedge_{i=1}^m C_i$ and (2) existence of a z-literal l in C_i and C_j, $i \neq j$, implies existence of an x-variable x such that $x \in C_i$ and $\neg x \in C_j$ or vice versa, and*
3. *closed under substitution of Σ_0^q formulas that contain only x-variables for x-variables.*

Then, GL^* is defined as follows.

Definition 2 *Given a proof of a sequent $\Gamma \rightarrow \Delta$, a variable is a parameter variable if it appears free in $\Gamma \rightarrow \Delta$.*

Definition 3 *GL^* is the propositional proof system G^* with cuts restricted to $\Sigma CNF(2)$ formulas in which no cut formula that is not Σ_0^q contains a non-parameter free variable.*

We chose this class of formulas because we are able to evaluate $\Sigma CNF(2)$ formulas, given an assignment to the free variables. Moreover, this problem is log space complete.

Lemma 1 ([9], Lemma 4.2.2). *Evaluating $\Sigma CNF(2)$ formulas is log space complete.*

This is an easy corollary to a theorem in [10], where Johannsen proved that the satisfiability problem for $CNF(2)$ formulas, $SAT(2)$, is log space complete. A $CNF(2)$ formula is a CNF formula where no variable appears more than twice. In Section 4, we will show how to find the assignment to the quantified variables in log space that satisfies the formula whenever possible.

The restriction on the free variables in cut formulas might seem unnatural, but it is necessary. Let H^* be the proof system G^* with cuts restricted to $\Sigma CNF(2)$ formulas and no restriction on the free variables. At first, we tried to show that H^* captured log space reasoning, but we found that it p-simulates G_1^* proofs of Σ_1^q formulas.

At a high level, H^* proves there exists an output to a circuit by proving that there exists an output to gate i, given the values for the inputs and the output of the first $i-1$ gates. This can be expressed as a $\Sigma CNF(2)$ formula. Then, with repeated cutting, we can prove there exists an output to the ciruit given the values of the inputs. Since H^* allows non-parameter variables in the cut formulas, the cute can be done. This is a problem since determining the output of a circuit is P-complete. Using this idea, we are able to prove the following theorem.

Theorem 1. *H^* p-simulates G_1^* for Σ_1^q formulas.*

A complete proof can be found in [9].

3 Propositional Translations

This section is motivated by results in [11]. In that paper, Cook showed that theorems of the equational theory PV can be translated into a family of tautologies that have polynomial size extended Frege proofs. We will show that theorems of Σ_0^B-rec can be translated into a family of tautologies that have polynomial size GL^* proofs. This tells us that GL^* captures log space reasoning in the same way extended Frege captures polynomial time reasoning.

In this paper, the theories will be two sorted. Numbers are the first sort. We use a, b, x, y, z as number variables, and they are intended to range over the natural numbers. Binary strings (or finite sets of numbers) are the second sort. We use X, Y, Z as string variables, and they are intended to be strings of 0's and 1's, with leading 0's removed.

The standard language is $\mathcal{L}_A^2 = [0, 1, +, \cdot, ||; =_1, =_2, ()]$. The constants, 0 and 1, and the binary functions, $+$ and $\cdot$, have their usual meaning. The final function $|X|$ returns to size of the string X (or the least upper bound of the finite set X). The predicates $=_1$ and $=_2$ are equality between numbers and string, respectively. The predicate $\leq$ is the usual inequality between two numbers. The final predicate is used to access the bits of X. So, $X(b)$ is true if the bth bit of X is 1, and false otherwise.

In two sorted bounded arithmetic, strings are the important sort. This is why we define classes of formula based on string quantifiers. The class Σ_0^B of formulas is the set of $\mathcal{L}_A^2$ formulas with no string quantifiers and all number quantifiers are bounded. That is, the number quantifiers are of the form $\exists x \leq b$. The class Σ_1^B of formulas is the class of formulas with an initial block of bounded existential string quantifiers followed by a Σ_0^B formula. A bounded string quantifier is of the form $\exists Z \leq b\phi$, which means $\exists Z, |Z| \leq b \wedge \phi$. The class Σ_1^1 is the same as Σ_1^B except the string quantifiers do not have to be bounded.

Given a Σ_1^B formula $\phi(\boldsymbol{x}, \boldsymbol{X})$ over the language $\mathcal{L}_A^2$, we want to translate it into a family of propositional formulas $||\phi(\boldsymbol{x}, \boldsymbol{X})||[\boldsymbol{n}]$, where the sizes of the formulas are bounded by a polynomial in $\boldsymbol{n}$ and the values assigned to $\boldsymbol{x}$. We use the translation described in [1, 7]. It is a modification of the Paris-Wilkie translation (see [1]).

The first step is to substitute constant values for all the number variables $\boldsymbol{x}$. For now we assume ϕ has no free number variables. The formula $||\phi(\boldsymbol{X})||[\boldsymbol{n}]$ is meant to be a formula that says $\phi(\boldsymbol{X})$ is true whenever $|X_i| = n_i$ for every string in $\boldsymbol{X}$ and the number variables are equal to the constants that replaced them. Then if $\phi(\boldsymbol{X})$ is true for all $\boldsymbol{X}$, then $||\phi(\boldsymbol{X})||[\boldsymbol{n}]$ is a tautology for all values for $\boldsymbol{n}$. Note that any term t that appears in ϕ can be evaluated immediately. This is because there are no number variables and the size of each string variable is known. So we will let $val(t(\boldsymbol{n}))$ be the value of the term. The variables $\boldsymbol{n}$ will often be omitted since they are understood. The free variables in the propositional formula will be $p_j^{X_i}$ for $j < n_i - 1$. The variable $p_j^{X_i}$ is meant to represent the value of the jth bit of X_i; we know that the n_ith bit is 1, and for $j > n_i$, we know the jth bit is 0. The definition proceeds by structural induction on ϕ.

Suppose ϕ is an atomic formula. Then it has one of the following forms: $s = t$, $s < t$, $X_i(t)$, or one of the trivial formulas $\bot$ and $\top$, for terms s and t. In the first case, we define $||\phi(\boldsymbol{X})||[\boldsymbol{n}]$ as the formula $\top$, if $val(s) = val(t)$, and $\bot$, otherwise. A similar construction is done for $s < t$. If ϕ is one of the trivial formulas, then $||\phi(\boldsymbol{X})||[\boldsymbol{n}]$ is the same trivial formula. So now suppose $\phi(\boldsymbol{X}) =_{syn} X_i(t)$. Let $j = val(t)$. Then the translation is defined as follows:

$$||\phi(\boldsymbol{X})||[\boldsymbol{n}] =_{syn} \begin{cases} p_j^{X_i} \text{ if } j < n_i - 1 \\ \top \text{ if } j = n_i - 1 \\ \bot \text{ if } j > n_i - 1 \end{cases}$$

Now for the inductive part of the definition. Suppose $\phi =_{syn} \alpha \wedge \beta$. Then

$$||\phi(\boldsymbol{X})||[\boldsymbol{n}] =_{syn} ||\alpha(\boldsymbol{X})||[\boldsymbol{n}] \wedge ||\beta(\boldsymbol{X})||[\boldsymbol{n}].$$

When the connective is $\vee$ or $\neg$, the definition is similar. Let $j = val(t)$. If the outer most connective is a quantifier, then the translation is defined as

$$||\exists y \leq t, \alpha(y, \boldsymbol{X})||[\boldsymbol{n}] =_{syn} \bigvee_{i=0}^{j} ||\alpha(i, \boldsymbol{X})||[\boldsymbol{n}]$$

$$||\forall y \leq t, \alpha(y, \boldsymbol{X})||[\boldsymbol{n}] =_{syn} \bigwedge_{i=0}^{j} ||\alpha(i, \boldsymbol{X})||[\boldsymbol{n}]$$

$$||\exists Y \leq t, \alpha(Y, \boldsymbol{X})||[\boldsymbol{n}] =_{syn} \exists p_0^Y, \ldots, \exists p_{j-2}^Y, \bigvee_{i=0}^{j} ||\alpha(Y, \boldsymbol{X})||[i, \boldsymbol{n}]$$

This completes the definition. Note that $\exists Y \leq b$ means there exists a string with size at most b.

3.1 The Theory $\boldsymbol{VL'}$

There have been a number of theories that capture log space reasoning. We will use Σ_0^B-rec [12]. The theory Σ_0^B-rec is axiomatized by the axioms of V^0

(explained below) plus the X-rec axiom below. This formula says that, if every vertex in the graph with nodes $\{0, \ldots, a\}$ and edge relation $X(i,j)$ has out-degree at least 1, then there exists a path of length b.

$$[\forall x \leq a \exists y \leq a X(x,y)] \supset \exists Z, \forall w \leq b \exists! x \leq a Z(w,x) \\ \wedge \forall w < b \forall x \leq a \forall y \leq a[Z(w,x) \wedge Z(w+1,y) \supset X(x,y)]. \tag{X-rec}$$

In this formula, $\exists! x\phi$ means "there exists a unique x such that ϕ is true." The theory V^0 has axioms that define addition, multiplication, the inequalities, and the size of a string. In addition, V^0 has the Σ_0^B-COMP axiom:

$$\exists Z \leq a \forall i < a, Z(i) \iff \phi(i),$$

where ϕ is a Σ_0^B formula that does not mention Z. V^0 is known to correspond to AC^0. See [1, 6] for details on V^0.

We know Σ_0^B-rec corresponds to log space because of the Σ_1^1-definability theorem below.

Theorem 2. *A function is Σ_1^1-definable in Σ_0^B-rec if and only if it is a log space function.*

See [12, 9] for a proof.

We want to reformulate the axioms of Σ_0^B-rec so they translate into $\Sigma CNF(2)$ formulas. With the exception of Σ_0^B-COMP, the axioms of V^0 are Σ_0^B, so they translate into Σ_0^q formulas, which are $\Sigma CNF(2)$. This means we only need to consider Σ_0^B-COMP and X-rec. We are not going to worry about Σ_0^B-COMP; we will handle this axiom the same way Cook and Morioka did in [7]. That is, if the proof system is asked to cut the translation of an instance of the Σ_0^B-COMP axiom, then the propositional proof is changed so that the cut becomes $\bigwedge_{i=0}^{t}[||\phi(i)|| \iff ||\phi(i)||]$, which is $\Sigma CNF(2)$. To take care of X-rec, we will define a new theory that is equivalent to Σ_0^B-rec by replacing the X-rec axiom.

Let Σ_0^B-edge-rec be the axiom scheme

$$\exists Z \leq 1 + \langle b, a, a\rangle[\rho_1 \wedge \rho_2 \wedge \rho_3 \wedge \rho_4 \wedge \rho_5 \wedge \rho_6 \wedge \rho_7 \wedge \rho_8],$$

where

$$\begin{aligned}
\rho_1 &=_{syn} \forall j < a, \neg Z(0,0,j) \vee \phi(0,j) \vee \exists l < j \phi(0,l)) \\
\rho_2 &=_{syn} \forall j \leq a \forall k < j, \neg Z(0,0,j) \vee \neg\phi(0,k) \vee \exists l < k\phi(0,l)) \\
\rho_3 &=_{syn} \forall i \leq a \forall j \leq a, i = 0 \vee \neg Z(0,i,j) \\
\rho_4 &=_{syn} \forall w < b \forall i \leq a \forall j \leq a, \neg Z(w+1,i,j) \\
&\qquad \vee \exists h \leq a Z(w,h,i) \vee \neg\phi(i,j) \vee \exists l < j\phi(i,l) \\
\rho_5 &=_{syn} \forall w < b \forall i \leq a \forall j < a, \neg Z(w+1,i,j) \vee \phi(i,j) \vee \exists l < j\phi(i,l) \\
\rho_6 &=_{syn} \forall w < b \forall i \leq a \forall j \leq a \forall k < j, \neg Z(w+1,i,j) \vee \neg\phi(i,k) \vee \exists l < k\phi(i,l) \\
\rho_7 &=_{syn} \exists i \leq a \exists j \leq a, Z(b,i,j) \\
\rho_8 &=_{syn} \forall \langle w,i,j\rangle \leq \langle b,a,a\rangle, [w > b \vee i > a \vee j > a] \supset \neg Z(w,i,j)
\end{aligned}$$

where $\phi(i, j)$ is a Σ_0^B formula that does not mention Z, but may have other free variables. Informally this axiom says there exists a string Z that gives a pseudo-path of length b in the graph with a nodes and edge relation $\phi(i, j)$. This path starts at node 0. If (i, j) is an edge in this path, then j is the smallest number with an edge from i to j, or $j = a$ when there are no outgoing edges. Note that the edge may not exists in the original graph when $j = a$. This is why we call is a pseudo-path. If (i, j) is the wth edge in the path, then $Z(w, i, j)$ is true, and $Z(w, i', j')$ is false for every other pair. It is not immediately obvious the axiom says this, so we will look at it closer.

Let Z be a string that witnesses the axiom. We want to make sure Z is the path described above. Looking at ρ_3, we see the path starts at 0. Suppose $Z(0, 0, j)$ is true. We must show that j is the first node adjacent to 0. This follows from ρ_1, which guarantees $\phi(i, j)$ is true when $j < a$, and ρ_2, which guarantees $\phi(i, k)$ is false when $k < j$. A similar argument can be made with ρ_5 and ρ_6 to show that every node is the smallest node adjacent to its predecessor. To make sure the path is long enough, we have ρ_7, which says there is a bth edge, and ρ_4, which says if there is a $(w + 1)$th edge there is a wth. As you may have noticed, there are parts of this formula that semantically are not needed. For example, the $\exists l < j\phi(0, l)$ in ρ_1 is not needed. It is used to make sure the axiom translates into a $\Sigma CNF(2)$ formula. We add ρ_8 to make sure there is a unique Z that witnesses this axiom.

Notation 2 *For simplicity, ψ_ϕ is the Σ_0^B part of the Σ_0^B-edge-rec axiom instantiated with ϕ. Note this includes the bound on the size of Z. So the axiom can be written as $\exists Z\psi_\phi$.*

So now we define the theory VL'.

Definition 4 *VL' is the theory axiomatized by the axioms of V^0, the Σ_0^B-edge-rec axioms, and Axiom (1). The language of VL' is the language of V^0 plus a string constant C with defining axiom*

$$|C| = 0 \tag{1}$$

We add the string constant to the language so we can put VL' proofs in free variable normal form (below). We do not use the constant for any other reason. Also, in the translation, we can treat C a string variable with $n = 0$.

In [9], the author proved that VL' and Σ_0^B-rec are equivalent. Since the X-rec and Σ_0^B-edge-rec axioms are semantically similar this is not too difficult, but to prove the Σ_0^B-edge-rec axiom from X-rec does require a trick used in [12] to get the path to start at 0. This gives us the following lemma.

Lemma 2. *The theory Σ_0^B-rec is equivalent to VL'.*

So now we know that VL' captures log space reasoning, and it does not capture reasoning for a larger complexity class.

The next step is to be sure the translation of this axiom is a $\Sigma CNF(2)$ formula. This is done by a careful inspection of the formula and is left to the reader.

Lemma 3. *The formula* $||\exists Z\psi_\phi(a,b,Z)||$ *is a* $\Sigma CNF(2)$ *formula.*

3.2 Cut Variable Normal Form

In this section, we want to find a normal form for VL' proofs that makes sure the translation of VL' proofs satisfy the variable restriction for GL^*. The normal form we want is *cut variable normal form* (CVNF) and is defined as follows.

Definition 5 *A formula* $\phi(Y)$ *is bit-dependent on* Y *if there is an atomic subformula of* ϕ *of the form* $Y(t)$*, for some term* t*.*

Definition 6 *A proof is in free variable normal form if every non-parameter free variable* y *or* Y *that appears in the proof is used as an eigenvariable of an inference exactly once, and every formula that contains* y *or* Y *appears before that inference.*

Definition 7 *A cut in a proof is anchored if the cut formula is an instance of an axiom.*

Definition 8 *A* VL' *proof* π *is in* cut variable normal form if π is (1) in free variable normal form, (2) every cut with a non-Σ_0^B cut formula is anchored, and (3) no cut formula that is an instance of the Σ_0^B-edge-rec axiom is bit-dependent on a non-parameter free string variable.

It is known how to find a proof with the first two properties [6, 5], but, to our knowledge, no property similar to the third has ever been considered.

The main theorem of this section is

Theorem 3. *Suppose* $VL' \vdash \exists Z \leq t\phi(Z)$ *for some* Σ_0^B *formula* ϕ*. Then there exists a* VL'*-proof* π *of* $\exists Z \leq t\phi(Z)$ *such that* π *is in CVNF.*

The proof of this theorem is the most technical in this paper. At a high level, it amounts to showing Σ_0^B-edge-rec is closed under substitution of strings defined by Σ_0^B-edge-rec and Σ_0^B-COMP. We begin with an anchored proof that is in free variable normal. We want to change every cut that violates condition (3) in the definition of CVNF. Consider the proof given in Figure 1. This is a

$$\dfrac{\dfrac{\begin{matrix} P \\ \vdots \\ \exists Z\psi_{\phi(Y)}(Z), \gamma(Y), \Gamma \to \Delta \end{matrix} \qquad \begin{matrix} \vdots \\ \gamma(Y), \Gamma \to \Delta, \exists Z\psi_{\phi(Y)}(Z) \end{matrix}}{\gamma(Y), \Gamma \to \Delta}}{\exists Y\gamma(Y), \Gamma \to \Delta}$$

Fig. 1. Example of a proof that is not in CVNF

simple example of what can go wrong. The general case is handled in the same way, so we will only consider this case.

Since all Σ_1^B cut formulas are anchored and the $\exists Y \gamma(Y)$ must eventually be cut, it is be an instance of Σ_0^B-COMP or Σ_0^B-edge-rec. The hard part is to prove the following lemma.

Lemma 4. *For any Σ_0^B formula $\phi(Y)$, there exist Σ_0^B formulas ϕ_1 and ϕ_2 such that ϕ_1 is not bit-dependent on Y and V^0 proves the sequent*

$$\gamma(Y), \psi_{\phi_1}(Z), \forall i < t[Z'(i) \iff \phi_2(Z)] \rightarrow \psi_{\phi(Y)}(Z').$$

Proof sketch. This proof is divided into two cases. In the first case, we assume

$$\gamma(Y) =_{syn} Y \leq t \forall i < t[Y(i) \iff \phi'(i)].$$

That is, it is an instance of Σ_0^B-COMP. We know Y must appear in that position because it gets quantified. In this case, ϕ_1 is ϕ with every atomic formula of the form $Y(s)$ replaced by $s < t \wedge \phi'(s)$, and ϕ_2 is not needed.

For the second case, we assume $\gamma(Y) =_{syn} \psi_{\phi'}(Y)$. In this case we use branching programs. We give a Σ_0^B description of a branching program BP_1 that computes Z' and a branching program BP_2 that computes Y. BP_1 at some point branches on Y. So we construct BP_3 by composing BP_1 and BP_2. Anytime BP_1 needs to branch on Y it runs BP_2 to see what it should do. The formula ϕ_1 is the Σ_0^B description of BP_3. The formula ϕ_2 represents the AC^0 function that extracts Z' from the run Z of BP_3. We use branching programs because log space can be defined in terms of branching programs.

Using this lemma, we are able to change the proof in Figure 1 into the proof in Figure 2. In that proof, P' is the proof P with the rules that introduced $\exists Z$ ignored, and Q is an anchored V^0 proof, which we know exists by the lemma above. This gives us a new proof of the same formula that still satisfies properties (1) and (2) in Definition 8 and it contains one less cut that is bit-dependent on Y, but it might be bit-dependent on different non-parameter variables. However, if we do things in the correct order, we can repeat the transformation and, eventually, we will get a proof that is in $CVNF$.

Using this manipulation, we prove Theorem 3.

Proof (Proof of Theorem 3). It would be nice to be able to simply say we can repeatedly apply the manipulations above and eventually the proof will be in CVNF, but this is not obvious. In the manipulation, if $\gamma(Y)$ is bit-dependent on a string variable other than Y, then the new Σ_0^B-edge-rec cut formula is bit-dependent on that variable. This includes non-parameter string variables. So we need to state our induction hypothesis more carefully.

Let $Y_1, \ldots, Y_n$ be all the non-parameter free string variables that appear in π ordered such that the variable Y_i is used as a eigenvariable before Y_j for $i < j$. This implies Y_i does not appear in $\gamma(Y_j)$ in the manipulations above. So now suppose no Σ_0^B-edge-rec cut formula is bit-dependent on the variables $Y_1, \ldots, Y_k$, for some $k < n$. Then we can manipulate π such that the same

$$\dfrac{\dfrac{\begin{array}{cc} P' & Q \\ \vdots & \vdots \\ \psi_{\phi(Y)}(Z), \gamma(Y), \Gamma \to \Delta & \gamma(Y), \psi_{\phi_1}(Z), \tau(Z') \to \Delta, \psi_{\phi(Y)}(Z) \end{array}}{\psi_{\phi_1}(Z), \tau(Z'), \gamma(Y), \Gamma \to \Delta}}{\psi_{\phi_1}(Z), \exists Z' \tau(Z'), \gamma(Y), \Gamma \to \Delta}$$

$$\dfrac{\dfrac{\psi_{\phi_1}(Z), \exists Z' \tau(Z'), \gamma(Y), \Gamma \to \Delta \qquad \to \exists Z' \tau(Z')}{\psi_{\phi_1}(Z), \gamma(Y), \Gamma \to \Delta}}{\exists Z \psi_{\phi_1}(Z), \gamma(Y), \Gamma \to \Delta}$$

$$\dfrac{\dfrac{\exists Z \psi_{\phi_1}(Z), \gamma(Y), \Gamma \to \Delta \qquad \to \exists Z \psi_{\phi_1}(Z)}{\gamma(Y), \Gamma \to \Delta}}{\exists Y \gamma(Y), \Gamma \to \Delta}$$

Fig. 2. Modification of the proof in Figure 1. The formula $\tau(Z')$ is used to replace $\forall i < t[Z'(i) \iff \phi_2(Z)]$

holds for the variables $Y_1, \ldots, Y_{k+1}$. To accomplish this, we simply manipulate every Σ_0^B-edge-rec cut formula that is bit-dependent on Y_{k+1} as described above. Since $Y_1, \ldots, Y_k$ cannot appear in $\gamma(Y_{k+1})$, those variables will not violate the condition. So by induction, we can get a proof that is in CVNF.

3.3 Translating Theorems of VL'

We are now prepared to prove the translation theorem. The proof is done by induction on the length of the proof. For the base case, we need to prove the translation of the axioms of VL'. Since every axiom except Σ_0^B-edge-rec is an axiom of VNC^1, we know those axioms have polynomial size G_0^* proofs [7] and, therefore, polynomial size GL^* proofs as well. Axiom (1) is easy to prove since it translates to $\to \top$. We still need to show how to prove the Σ_0^B-edge-rec axiom in GL^*. Recall that we write the axiom as $\exists Z \psi_\phi(a, b, Z)$. Note that the axiom does have a bound on Z, but it has been omitted since the specific bound is not important.

Lemma 5. *The formula $||\exists Z \psi_\phi(a, b, Z)||$ has a GL^* proof of size $p(a, b)$ for some polynomial p.*

Proof sketch. The proof is done by a brute force induction. We prove, in GL^*, that, if there exists a pseudo-path of length b, then there exists a pseudo-path of length $b + 1$. It is easy to prove there exists a pseudo-path of length 0. With repeated cutting we get our final result. The entire path is quantified, so we do not cut non-parameter free variables.

We now prove the main theorem of this section.

Theorem 4. *Suppose VL' proves $\exists Z < t\phi(\boldsymbol{x}, \boldsymbol{X}, Z)$. Then there are polynomial size GL^* proofs of $||\exists Z < t\phi(\boldsymbol{x}, \boldsymbol{X}, Z)||[\boldsymbol{n}]$.*

Proof. By Theorem 3, there exists a VL' proof π of $\exists Z < t\phi(\boldsymbol{x}, \boldsymbol{X}, Z)$ that is in CVNF.

We proceed by induction on the depth of π. The base case follows from Lemma 5 and the comments that precede it. The inductive step is divided into cases: one for each rule. With the exception of cut, every rule can be handled the same way it is handled in the $V^1 - G_1^*$ Translation Theorem [6], and will not be repeated here.

When looking at the cut rule, there are three cases. If the cut formula is Σ_0^B, then we simply cut the corresponding Σ_0^q formula in the GL^* proof. If the cut formula is not Σ_0^B, then it must be anchored since the proof is in CVNF. This means the cut formula is an instance of Σ_0^B-edge-rec or an instance of Σ_0^B-COMP. First suppose it is an instance of Σ_0^B-edge-rec. Then we are able to cut the corresponding formula in the GL^* proof. This is because the axiom translates into a $\Sigma CNF(2)$ formula, and the free variables in the translation are parameter variables since the formula is not bit-dependent on non-parameter string variables,

When the cut formula is an instance of Σ_0^B-COMP, we apply the same transformation as in the proof of the $VNC^1 - G_0^*$ translation theorem [7]. That is, we remove the quantifiers by replacing the variables with Σ_0^q formulas that witness the quantifiers. This change does not effect other cuts since their free variables are parameter variables or they are Σ_0^q formulas and remain Σ_0^q after the substitution. The current cut formula becomes a Σ_0^q formula, which can be cut.

4 Proving GL^* Is Sound in Σ_0^B-rec

In this section, we show that GL^* does not capture reasoning for a higher complexity class. This is done by proving, in Σ_0^B-rec, that GL^* is sound. This idea comes from [11] where Cook showed that PV proves extended Frege is sound and [2] where Krajicek and Pudlak showed T_2^i proves G_i is sound for $i > 0$.

We will actually show that $\overline{VL}$ proves GL^* is sound. This theory is a universal conservative extension of Σ_0^B-rec. In [9], the author proved that $\overline{VL}$ proves induction on Σ_0^B formulas that contain log space functions. Formally, this is $\Sigma_0^B(\mathcal{L}_{FL})$-IND. In general, we will give informal proofs, but we will be sure induction hypotheses do not use functions that are not log space. Once the proof in $\overline{VL}$ is done, we will know that it can be done in Σ_0^B-rec as well since $\overline{VL}$ is a conservative extension of Σ_0^B-rec [9].

We start the proof by giving a log space algorithm that witnesses $\Sigma CNF(2)$ formulas when the formula is true. This algorithm is the algorithm given in [10] with a few additions to find the satisfying assignment. This algorithm can be formalized in $\overline{VL}$ since it is a log space algorithm, and $\overline{VL}$ proves that it is correct. This means we can use this function in induction hypotheses. We then use this function to define a log space function that witnesses GL^* proofs. We finish by proving in $\overline{VL}$ that the algorithm is correct.

4.1 Witnessing $\Sigma CNF(2)$ Formulas

We want to define a function W that takes as input a $\Sigma CNF(2)$ formula and an assignment to the free variables, and returns an assignment to the quantified variables that satisfies the quantifier free portion of the formula whenever possible. Let $\exists \boldsymbol{z} F(\boldsymbol{z}, \boldsymbol{x})$ be a $\Sigma CNF(2)$ formula, where F is quantifier free, and let $\boldsymbol{v}$ be the values assigned to $\boldsymbol{x}$. We begin by using $\boldsymbol{v}$ to simplify F. By the form of $\Sigma CNF(2)$ formulas, the simplified formula is $CNF(2)$. We now need to find a satisfying assignment to the simplified formula, if one exists. The simplified formula will also be called F.

To find the satisfying assignment, we construct an undirected tagged graph (G, T) based on F. This is done as in [10]. The graph G has a vertex v_i for every clause C_i in F. There is an edge between two vertices v_i and v_j if there is a literal l such that l is in C_i and $\overline{l}$ is in C_j. A vertex is tagged if the corresponding clause contains a pure literal.

Based on this construction, Johannsen proved the following lemma [10].

Lemma 6. *F is satisfiable if and only if it is possible to direct the edges of G such that there are no untagged sinks.*

The proof of this lemma can be done in $\overline{VL}$ since the direction for the edges can be easily construction from the satisfying assignment and vice versa. Johannsen noted that the edges can be properly directed if and only if G does not contain an untagged tree. The only if direction is easy to prove using a counting argument. A tree has fewer edges than nodes, so directing the edges cannot make every node a non-sink. This can be formalized in $\overline{VL}$ since $\overline{VL}$ extends VTC^0, and, therefore, proves the pigeon hole principle for $\Sigma_0^B(\mathcal{L}_{FL})$ formulas [9]. To prove the other direction in $\overline{VL}$, we give a log space algorithm that directs the edges appropriately, and prove the correctness of the algorithm in $\overline{VL}$. This also gives us the function W we are looking for.

We use a trick first used in [13] to find a cycle in a graph. For each edge $e = (u, v)$ in G, we call its two end points e^u and e^v, with the obvious meaning. Given an end point p, we the edge can be obtain by $e(p)$ and the vertex by $v(p)$. Then we can define the permutations σ_G, which is the product of the transposition $(e^u e^v)$ for every edge in G, and ρ_G, which is the product of the cycles $(e_1^v \dots e_n^v)$ for each vertex v, where $e_1, \dots, e_n$ are the edge incident to v. Then the permutation $\pi_G = \rho_G \circ \sigma_G$. We will often talk about the graph of π_G, which will also be called π_G. This permutation is useful because of the following nice property first proved in [13].

Definition 9 *An end point e^u of a vertex $e = (u, v)$ is trivially traversed if the there are no end points e_1^u on the path from e^u to e^v in π_G. If the path from e^u to e^v does not exists, then e^u is not trivially traversed.*[1]

[1] This definition comes from a personal communication from Mark Braverman.

Lemma 7. *The connected component of G that contains edge $e = (u, v)$ is a tree if and only if every end point in the cycle of π_G that contains e^u is trivially traversed. Moreover, this is provable in $\overline{VL}$.*

This gives us a method of finding cycles.

The algorithm to find appropriate directions for the edges is done in stages. In stage i, we consider the ith vertex v_i. If the vertex is tagged, we continue. Otherwise, we search G, using π_G, for a cycle or tagged vertex that can be reached from v_i. If no cycle or tagged vertex is found, then we found an untagged tree and we can stop. The edges on the path from v_i to the cycle or tagged vertex are directed away from v_i. The edges in the cycle are directed to form a directed cycle, which direction does not matter. If one of these edges was given a direction in an earlier stage, that direction is overwritten with the new value.

If we let $FindTagOrCycle(G, p)$ be a function that returns the closest end point q to p in π_G such that $v(q)$ is tagged or q is not trivially traversed. (This can be done using the algorithm described in [10].) Then the algorithm described above can be implemented as in Algorithm 1.

```
for i = 1...n do
  if v_i is not tagged then
    w = FindTagOrCycle(G, e^{v_i}) for some edge e adjacent to v_i.
    If w = null, stop.
    Else, w' = e^{v_i}
    while w' ≠ w do
      Direct e(w') away from v(w')
      w' = π_G(w')
    end while
  end if
end for
```

Algorithm 1: Algorithm to direct edges on a tagged graph

The correctness of the algorithm follows from the following invariant: After stage i, the vertices $v_1, \ldots, v_i$ are tagged or are not sinks. It is easy to check that this holds.

We claim this algorithm can be done in log space. Note that an edge may be directed multiple times, but the values are never used. So, to determine the direction of an edge e, we can run the algorithm, but only keep track of e's direction. The cycles and tagged vertex are found by searching π_G, which has out-degree 1. Therefore the search can be done in log space. Note that we are using that log space is closed under composition.

The final thing to do is prove the correctness of the algorithm in $\overline{VL}$. This can be done by induction on the invariant above. Since the algorithm is log space, the invariant can be stated as a $\Sigma_0^B(\mathcal{L}_{FL})$ formula, so the induction can be done in $\overline{VL}$.

The function W can be defined by applying the reduction to the input and output of this algorithm. We add that pure literals are assigned $\top$ and variables that are still not assigned a value are assigned $\bot$.

4.2 Witnessing GL^* Proofs

Let π be a GL^* proof of a Σ_1^q formula $\exists \boldsymbol{z} P(\boldsymbol{x}, \boldsymbol{z})$, and let A be an assignment to the parameter variables (Definition 2). We will assume π is in free variable normal form (Definition 6). If it is not, we can rename variables to put it in free variable normal form. The renaming can be done in log space since all that is really required is to traverse the proof, which is a tree, to determine an appropriate name.

Let $\Gamma_i \rightarrow \Delta_i$ be the ith sequent in π. To prove the soundness of GL^*, we define a function $Wit(i, \pi, A)$ that will find a formula in Γ_i that is false or a formula in Δ_i that is true. We will prove by induction that for any assignment to all of the free variables if Γ_i and Δ_i, $Wit(i, \pi)$ will find at least one formula that satisfies the sequent.

There are two things to note. Every formula in Γ_i is $\Sigma CNF(2)$, which means it can be evaluated. Also, we need an assignment that gives appropriate values to the non-parameter free variables that could appear. To take care of this second point, we extend A to an assignment A' as follows:

1: Given a non-parameter free variable y, find the $\exists$-left inference in π that uses y as an eigenvariable. Let z be the new bound variable and let F be the principal formula.
2: Find the descendant of F that is used as a cut formula. Let F' be the cut formula. Note that F is a subformula of F', and, because of the variable restriction on cut formulas, every free variable in F' is a parameter variable.
3: Assign y the value that $W(F', A)$ assigns z.

The reason for this particular assignment will become evident in the proof of Lemma 8.

We can now define $Wit(i, \pi, A')$, which witnesses $\Gamma_i \rightarrow \Delta_i$. Wit will go through each formula in the sequent to find a formula that satisfies the sequent. $\Sigma CNF(2)$ formula are evaluated using the algorithm described in the previous section. We will not focus our attention on other Σ_1^q formulas, which must appear in Δ_i. Each Σ_1^q formula $F =_{syn} \exists \boldsymbol{z} F^*(\boldsymbol{z})$ in Δ is evaluated by finding a witness to the quantifiers as follows:

1: Find a formula F' in π that is a ancestor of F, is satisfied by A', and is a Σ_0^q formula of the form $F^*(z_1/B_1, \ldots, z_n/B_n)$, where each B_i is Σ_0^q
2: z_i is assigned $\top$ if A' satisfies B_i, otherwise it is assigned $\bot$
3: if no such F' exists, then every bound variable is assigned $\bot$.

Lemma 8. *For every sequent $\Gamma_i \rightarrow \Delta_i$ in π, $Wit(i, \pi, A')$ finds a false formula in Γ_i or a true formula in Δ_i.*

Proof. We prove the theorem by induction on the depth of the sequent. For the base case, the sequent is an axiom, and the theorem obviously holds. For the inductive step, we need to look at each rule. We can ignore $\forall$-left and $\forall$-right since universal quantifiers do not appear in π.

We will not assume all formulas in Γ_i are true and all $\Sigma CNF(2)$ formulas in Δ_i as false. So we need to find a Σ_1^q formula in Δ_i that is true.

Consider cut. Suppose the inference is

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

First suppose F is true. By induction, with the upper left sequent, Wit witnesses one of the formulas in Δ. Then the corresponding formula in the bottom sequent is witnessed by Wit. This is because the ancestor of the formula in the upper sequent that gives the witness is also an ancestor of the corresponding formula in the lower sequent. If F is false, it cannot be the formula that was witnessed in the upper right sequent, and a similar argument can be made.

Consider $\exists$-right. Suppose the inference is

$$\frac{\Gamma \to \Delta, F(B)}{\Gamma \to \Delta, \exists z F(z)}$$

First suppose $F(B)$ is Σ_0^q. If it is false, we can apply the inductive hypothesis, and, by an argument similar to the previous case, prove one of the formulas in Δ must be witnessed. If $F(B)$ is true, then Wit will witness $\exists z F(z)$ since $F(B)$ is the ancestor that gives the witness. If $F(B)$ is not Σ_0^q, then we can apply the inductive hypothesis, and, by the same argument, find a formula that is witnessed.

The last rule we will look at is $\exists$-left. Suppose the inference is

$$\frac{F(y), \Gamma \to \Delta}{\exists z F(z), \Gamma \to \Delta}$$

To be able to apply the inductive hypothesis, we need to be sure that $F(y)$ is satisfied. If $\exists z F(z)$ it true, then we know $F(y)$ is satisfied by the construction of A': the value assigned to y is chosen to satisfy $F(y)$ if it is possible. Otherwise, $\exists z F(z)$ is false, and we do not need induction.

For the other rules the inductive hypothesis can be applied directly and the witness found as in the previous cases.

Theorem 5. *$\overline{VL}$ proves GL^* is sound for proofs of Σ_1^q formulas.*

Proof. The functions W and Wit are log space functions and can be formalized in $\overline{VL}$. A function that finds A', given A, can also be formalized since it is log space. The final thing to note is that the proof of Lemma 8 can be formalized in $\overline{VL}$ since the induction hypothesis can be express as a $\Sigma_0^B(\mathcal{L}_{FL})$ formula and the induction carried out.

The reason this proof does not work for a larger proof system, say G_1^*, is because W cannot be formalized for the larger class of cut formulas. Also, if the variable restriction was not present, we would not be able to find A' in log space, and the proof would, once again, break down.

5 Concluding Remarks

To summarize, we have a proof system that corresponds to log space reasoning. This is a formula-based proof system, and it corresponds to Σ_0^B-rec in the usual way. This treatment of the proof system suggests a proof system for NL, which would be based on $2-SAT$ instead of $SAT(2)$.

One drawback with GL^* is the variable restriction. It forced us to prove a normal form for VL' proofs. This normal form is specific to VL', and this proof would have to be completely redone for other theories. On the other hand, the proof that GL^* is sound can be easily changed to work for other theories. All that really needs to be changed for the case of NL is the definition of the function W.

I would like to thank my supervisor Stephen Cook for providing many useful comments.

References

1. Cook, S.: Theories for complexity classes and their propositional translations. In Krajicek, J., ed.: Complexity of computations and proofs. Quaderni di Matematica (2003) 175–227. Also available at *http://www.cs.toronto.edu/˜sacook/*
2. Krajicek, J., Pudlak, P.: Quantified propositional calculi and fragments of bounded arithmetic. Zeitschrift f. Mathematikal Logik u. Grundlagen d. Mathematik **36** (1990) 29–46
3. Cook, S.: A survey of complexity classes and their associated propositional proof systems and theories, and a proof system for log space. Available at *http://www.cs.toronto.edu/˜sacook/* (2001)
4. Pollett, C.: A propositional proof system for R_2^i. In Beame, P.W., Buss, S.R., eds.: Proof Complexity and Feasible Arithmetics. Volume 39 of DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. AMS (1997) 253–278
5. Buss, S.R.: Introduction to proof theory. In Buss, S.R., ed.: Handbook of Proof Theory. Elsevier Science Publishers, Amsterdam (1998) 1–78
6. Cook, S.: CSC 2429s: Proof complexity and bounded arithmetic. Course notes. Available at *http://www.cs.toronto.edu/˜sacook/csc2429h.02* (2002)
7. Cook, S., Morioka, T.: Quantified propositional calculus and a second-order theory for NC^1. (Accepted for Archive for Math. Logic)
8. Krajicek, J.: Bounded Arithmetic, Propositional Logic, and Complexity Theory. Cambridge University Press (1995)
9. Perron, S.: GL^*: A propositional proof system for logspace. Master's thesis, University Of Toronto (2005). Available at *http://www.cs.toronto.edu/˜sperron/*
10. Johannsen, J.: Satisfiability problem complete for deterministic logarithmic space. In: STACS 2004, 21st Annual Symposium on Theoretical Aspects of Computer Science, Proceedings, Springer (2004) 317–325
11. Cook, S.A.: Feasibly constructive proofs and the propositional calculus. In: Proceedings of the 7-th ACM Symposium on the Theory of computation. (1975) 83–97
12. Zambella, D.: End extensions of models of linearly bounded arithmetic. Annals of Pure and Applied Logic **88** (1997) 263–277
13. Cook, S.A., McKenzie, P.: Problems complete for deterministic logarithmic space. Journal of Algorithms **8** (1987) 385–394