Relating the PSPACE reasoning power of Boolean Programs and Quantified Boolean Formulas

by

Alan Skelley

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Relating the PSPACE reasoning power of Boolean Programs and Quantified Boolean Formulas

Alan Skelley

Master of Science

Graduate Department of Computer Science

University of Toronto

2000

We present a new propositional proof system based on a recent new characterization of polynomial space (PSPACE) called Boolean Programs, due to Cook and Soltys. We show that this new system, BPLK, is polynomially equivalent to the system G, which is based on the familiar and very different quantified Boolean formula (QBF) characterization of PSPACE due to Stockmeyer and Meyer. We conclude with a discussion of some closely related open problems and their implications.

Acknowledgements

Thanks to my parents for being not so bad after all. A nod to NSERC for greasing the wheels with PGSA-208264-1998. My office-mates Steve “Stevenator” Myers, Iannis “Axiom” Tourlakis, John “Moonman” Watkinson, Jonathan “Animal” Shekter, Natasa “Stash” Przulj and Eric “Do. J” Joanis for many helpful discussions and productive distractions. Kleoni Ioannidou for moral support.

Tsuyoshi Morioka for helping with some production details. Michael Soltys, for the topic. My second reader, Toniann Pitassi, for reading under duress.

Especially thanks to my supervisor, Stephen Cook, for countless helpful discussions and crucial ideas, not to mention a lot of reading and correcting.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

We often argue that a particular mathematical concept is important if it is natural, which means that it surfaces in many places with different origins and definitions, and robust, such that a variety of disparate formulations of it end up being equivalent or at least closely related. Likewise, the applicability, maturity, and importance of a body of results are greater when that field is found to have a strong connection to another. Three areas of study intricately connected in such a useful way are computational complexity, the proof theory of arithmetic and propositional proof complexity.

Computational complexity is the study of computation and the resources required to perform it. A staggering number of different kinds of computation all fall into the domain of this field. It has practical aspects, directly impacting how real computations are done by real computers, and yet seemingly fundamental, easily explained problems remain unsolved despite a good deal of effort. A particularly glaring example is the famous P vs NP problem, which asks if those two classes of problems are equal. Starting from the NP-completeness results of Cook [12] the pressure mounted with no relief, leading even to detailed, formal analysis of known proof techniques and why they are all ineffectual at tackling such problems [25]. Many complexity classes are studied and conjectures about separations and hierarchies abound, yet results are elusive.

A different way of studying computational complexity is indirectly through logic. Many connections between the fields are known: complexity classes can be characterized as those sets or functions definable in certain theories; sets of models of formulas can be seen as languages or classes of languages; predicates or functions from certain complexity classes can be used to define new logics. A relevant example comprises the hierarchies of theories of bounded arithmetic T_2^i and S_2^i of Buss [4]. As shown in [6], [28] and [22], this bounded arithmetic hierarchy collapses if and only if S_2 proves that the polynomial hierarchy collapses.

Now, due to Cook [13], there is a translation from formulas of bounded arithmetic to polynomial-sized families of propositional formulas. Furthermore, if the bounded arithmetic formula has a proof in Cook's system PV (corresponding to polynomial-time reasoning), then its translations have polynomial-sized extended Frege proofs which can be found in polynomial time. We can replace PV by S_2^1 in the previous statement due to both theories robustly defining polynomial-time reasoning, though in different ways. Other translations are known and in particular there is a similar connection between T_2^i and G_i, and another between S_2^i and G_i^*, both due to [21]. There is another correspondence [23] between U_2^1 (a second order system of Buss') and G, although only for first-order, Σ_1^b formulas. In all of these latter correspondences, it is also the case that the bounded arithmetic system can prove reflection principles for, and thus simulate, the propositional system.

The full circle back to computational complexity is completed with the work of Cook and Reckhow in [10] and [14]. They show that P=co-NP if and only if there exists a polynomially bounded proof system, and additionally introduce many of the important definitions in the area such as those of proof systems, polynomial simulations, and so on. These results drive the study of propositional proof complexity and the search for lower bounds on propositional proof systems. Fine examples are the superpolynomial lower bounds for resolution, due to Haken [16] and bounded depth Frege systems, due to Ajtai [1]. For many seemingly stronger systems, however, no such results are known.

1.2 Overview of Thesis

The idea suggesting the results in this thesis is yet another connection between computational complexity and propositional proof complexity. When formulated in a Gentzen sequent style, many known propositional proof systems can be seen to be very similar, with the only difference between them being the computational power of what can be written at each line of the proof (or alternatively, what is allowed in the **cut** rule). Examples are Boolean formulas in Frege systems, single literals in resolution, Boolean circuits in extended Frege systems. Another example is the system G, which is a sequent-based system where formulas in the sequents are quantified boolean formulas (QBFs). These formulas have propositional variables and also propositional quantifiers. In this case, then, since evaluating QBFs is PSPACE-complete, the computational power which can be harnessed in sequents is PSPACE. We can restrict G to G_i by restricting the number of alternations of quantifiers allowed in the formulas, and the reasoning power is then that of Σ_i^p predicates.

Boolean programs were introduced by Cook and Soltys in [11]. A Boolean program defines a sequence of Boolean function symbols, where each function symbol is defined using a boolean formula which can include, in addition to the arguments to the function, invocations of the previously defined symbols. The authors of that paper showed that the problem of evaluating an invocation of a function symbol defined in this way, given the inputs and the Boolean program, is PSPACE complete. The question that then arises is whether a proof system formulated around Boolean programs would be equivalent to G. For this to occur, not only would Boolean programs and quantified Boolean formulas need to characterize the same complexity class, but there would need to be an effective way of translating between the two.

This thesis answers that question in the affirmative. After reviewing basic terminology and notation in chapter 2, in chapter 3 we define our new system BPLK in a straightforward way to take advantage of the expressive power of Boolean programs. In that chapter we also prove some basic results about the two systems in consideration.

Chapter 4 contains the first of the main results, which is a polynomial simulation of G by BPLK. We first show how to translate sequents from the language of G into equivalent sequents in the language of Boolean programs. As we discuss, the translation is not merely the Skolemization one might expect but rather something more sophisticated and reminiscent of Hilbert's ϵ-calculus. Following that we show how to simulate G by translating a proof in that system, line-by-line, into the language of Boolean programs and then filling in the gaps to make the result a proof in BPLK.

Chapter 5 presents the converse simulation. The translation used here first takes a Boolean program to a single formula which may be used to simultaneously evaluate all functions defined by that program. This formula is used to evaluate function symbols occurring in the original BPLK-proof and yields a translation of sequents. As in the last chapter, a line-by-line translation followed by some filling in of gaps gives the desired result.

Concluding, in chapter 6 we discuss some open problems and other issues raised by these results.

Chapter 2

Preliminaries

In this chapter we present some formal background about proof systems, which we first formally define. We present Gentzen's popular sequent-based system LK, which is the foundation for the two proof systems compared in this thesis, and also discuss quantified propositional logic and the system G and its subsystems. Finally, we comment on some notational conventions which we shall use.

2.1 Propositional Proof Systems

We shall consider a language consisting of the complete basis $\{\neg, \vee, \wedge\}$, parentheses, constants 0 (for false) and 1, and an infinite supply of atom symbols which we shall represent with a variety of lower-case letters. In the standard way, well-formed formulas in this language define truth functions or equivalently, Boolean functions, of the truth values of the atoms. TAUT is the set of propositional tautologies, formulas which evaluate to true on every assignment.

Definition 2.1.1. *A* proof system *P for a set S is a surjective polynomial-time computable function $P : \Sigma^* \to S$ for some alphabet Σ.*

We are interested in proof systems for TAUT. A P-proof of a tautology τ is a string

π such that $P(\pi) = \tau$. We denote by $|\pi|$ the number of symbols in π. We have the following important notion which allows us to compare the power of proof systems:

Definition 2.1.2. *If P and Q are proof systems, we say that P* polynomially simulates *(p-simulates) Q and write $P \leq_p Q$ if there is a polynomial-time computable function g such that for every string x, $P(g(x)) = Q(x)$.*

2.2 LK and Quantified Propositional Logic

A popular proof system is Gentzen's sequent system LK. LK is actually a proof system for predicate logic but we shall consider only the propositional fragment. Each line of an LK-proof is a sequent, a string of the form $\Gamma \longrightarrow \Delta$, where Γ and Δ are possibly empty finite sequences of propositional formulas. A sequent is satisfied if and only if either one of the formulas on the left (the *antecedent*) is falsified, or one of the formulas on the right (the *succedent*) is satisfied. Each sequent in a proof is either an initial sequent of the form $0 \longrightarrow$, $\longrightarrow 1$ or $a \longrightarrow a$ for an atom a, or it is derived from previous ones (its hypotheses) via one of the following inference rules (this set is the same as in [9], which is a slight modification of the ones in [20]):

weakening:

$$\textbf{left} \quad \frac{\Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, A}$$

exchange:

$$\textbf{left} \quad \frac{\Gamma_1, A, B, \Gamma_2 \longrightarrow \Delta}{\Gamma_1, B, A, \Gamma_2 \longrightarrow \Delta} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \longrightarrow \Delta_1, A, B, \Delta_2}{\Gamma \longrightarrow \Delta_1, B, A, \Delta_2}$$

contraction:

$$\textbf{left} \quad \frac{\Gamma_1, A, A, \Gamma_2 \longrightarrow \Delta}{\Gamma_1, A, \Gamma_2 \longrightarrow \Delta} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \longrightarrow \Delta_1, A, A, \Delta_2}{\Gamma \longrightarrow \Delta_1, A, \Delta_2}$$

$\neg$: introduction:

$$\textbf{left} \quad \frac{\Gamma \longrightarrow \Delta, A}{\neg A, \Gamma \longrightarrow \Delta} \qquad \text{and} \qquad \textbf{right} \quad \frac{A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, \neg A}$$

$\wedge$: introduction:

$$\textbf{left} \quad \frac{A, B, \Gamma \longrightarrow \Delta}{A \wedge B, \Gamma \longrightarrow \Delta} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \longrightarrow \Delta, A \quad \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, A \wedge B}$$

$\vee$: introduction:

$$\textbf{left} \quad \frac{A, \Gamma \longrightarrow \Delta \quad B, \Gamma \longrightarrow \Delta}{A \vee B, \Gamma, \longrightarrow \Delta} \qquad \text{and} \qquad \textbf{right} \quad \frac{\Gamma \longrightarrow \Delta, A, B}{\Gamma \longrightarrow \Delta, A \vee B}$$

cut:

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

Quantified propositional logic is what results when we add propositional quantifiers to our language. The semantics of $\forall x \phi(x, \overline{p})$ is that this formula is satisfied by a particular assignment if and only if $\phi(0, \overline{p}) \wedge \phi(1, \overline{p})$ is. Likewise the truth value of $\exists x \phi(x, \overline{p})$ is the same as that of $\phi(0, \overline{p}) \vee \phi(1, \overline{p})$. As in [7], when in the context of quantified propositional logic we shall divide variables into *bound variables* and *free variables*. Free variables may occur free in formulas and semiformulas, but may never be quantified. Bound variables may occur freely in semiformulas, and may be quantified in formulas and semiformulas. Sequents are constructed from formulas.

Additionally, we can define a hierarchy of quantified Boolean semiformulas. The following is a slight adaptation of the definition in [20]:

Definition 2.2.1. *The classes Π_i^q and Σ_i^q are defined as follows:*

1. $\Sigma_0^q = \Pi_0^q$ *are the quantifier-free propositional semiformulas.*

2. *If ϕ is Σ_i^q or Π_i^q then it is also Σ_j^q and Π_j^q for all $j > i$.*

3. *If* $\phi(x)$ *is* Σ_i^q *then* $\forall x \phi(x)$ *is* Π_{i+1}^q.

4. *If* $\phi(x)$ *is* Π_i^q *then* $\exists x \phi(x)$ *is* Σ_{i+1}^q.

5. *If* ϕ *is* Σ_i^q *(*Π_i^q*) then* $\neg\phi$ *is* Π_i^q *(*Σ_i^q *respectively).*

6. Σ_i^q *and* Π_i^q *are closed under* $\vee$ *and* $\wedge$.

7. Σ_i^q *(*Π_i^q*) is closed under existential (universal) quantification.*

Now, the proof system G is obtained by augmenting the set of inference rules of LK with the following:

$\forall$: introduction:

$$\textbf{left}\quad \frac{A(B), \Gamma \longrightarrow \Delta}{\forall x A(x), \Gamma \longrightarrow \Delta} \qquad \text{and} \qquad \textbf{right}\quad \frac{\Gamma \longrightarrow \Delta, A(p)}{\Gamma \longrightarrow \Delta, \forall x A(x)}$$

$\exists$: introduction:

$$\textbf{left}\quad \frac{A(p), \Gamma \longrightarrow \Delta}{\exists x A(x), \Gamma \longrightarrow \Delta} \qquad \text{and} \qquad \textbf{right}\quad \frac{\Gamma \longrightarrow \Delta, A(B)}{\Gamma \longrightarrow \Delta, \exists x A(x)}$$

where B is any formula and the atom p replaced does not occur in the conclusion of the corresponding inference. G_i is G with the restriction that all formulas appearing in a proof must be Σ_i^q or Π_i^q. It should be noted that although G and its subsystems derive tautological statements of quantified propositional logic, in this thesis we will consider them only as proof systems for propositional tautologies.

2.3 Boolean Programs

Boolean programs were introduced in [11] and are a way of specifying Boolean functions. It seems that perhaps representations can be remarkably (even exponentially) much shorter than with Boolean formulas or circuits, although a formal proof would be a breakthrough. Boolean programs are something like a generalization of the technique of using new atoms to replace part of a Boolean formula, which idea is the basis of extended Frege systems. The following definition is from that paper:

Definition 2.3.1 (Cook-Soltys). *A* Boolean Program *P is specified by a finite sequence $\{f_1, ..., f_m\}$ of function symbols, where each symbol f_i has an associated arity k_i, and an associated defining equation*

$$f_i(\overline{p_i}) := A_i$$

where $\overline{p_i}$ is a list $p_1, ..., p_{k_i}$ of variables and A_i is a formula all of whose variables are among $\overline{p_i}$ and all of whose function symbols are among $f_1, ..., f_{i-1}$. In this context the definition of a formula *is:*

1. *0,1, and p are formulas, for any variable p.*

2. *If f is a k-ary function symbol in P and $B_1, ..., B_k$ are formulas, then $f(B_1, ..., B_k)$ is a formula.*

3. *If A and B are formulas, then $(A \wedge B), (A \vee B)$ and $\neg A$ are formulas.*

The semantics are as for propositional formulas, except that when evaluating an application $f_i(\overline{\phi})$ of a function symbol, the value is defined, using the defining equation, to be $A_i(\overline{\phi})$. There is no free/bound distinction between variables in the language of Boolean programs.

An interesting property of Boolean programs which demonstrates their comparability to quantified Boolean formulas is the following theorem from [11]:

Theorem 2.3.2 (Cook-Soltys). *A Language L is in PSPACE iff L is computed by some uniform polynomial size family of Boolean programs.*

2.4 Notational Conventions

We shall use the following conventions of notation: Lower case English letters will represent atoms, with $x, y, z, ...$ reserved for bound variables, and with the further exception of $f, g, h, ...$ to be used for function symbols. Capital letters and lower case Greek letters will be used for formulas. An overline indicates a list: $\overline{a}$ is a list of variables $(a_1, ...)$ and $\overline{\overline{A}}$ is a list of lists of formulas $(\overline{A_1} = \{A_{1,1}, A_{1,2}, ...\}, \overline{A_2}, ...)$. A formula A may have free variables $\overline{p}$, and when we wish to emphasize that fact we shall write $A(\overline{p})$, although we may not explicitly display all free variables of A. $A(\overline{\phi})$ denotes the result of substituting the list of formulas ϕ for the free variables of A. Since we have separated bound and free variables, in the quantified case we are automatically assured that $\overline{\phi}$ is free for $\overline{p}$ in $A(\overline{p})$, which is to say that no free variables of $\overline{\phi}$ will end up bound by any of A's quantifiers in the substitution.

We shall use the following symbols:

Definition 2.4.1 (=,⊃,≡). *The symbols '$=$' and '$\supset$' are not in the language of either system we consider, but we shall use them as abbreviations. $\overline{A} = \overline{B}$ abbreviates*

$$((\neg A_1 \vee B_1) \wedge (\neg B_1 \vee A_1) \wedge ... \wedge (\neg A_k \vee B_k) \wedge (\neg B_k \vee A_k)).$$

$A \supset B$ abbreviates $\neg A \vee B$. The symbol '$\equiv$' will be used to denote syntactic equality.

Finally, consider that although in general not all proof systems need be of this form, we shall consider only systems where a proof consists of a sequence of lines, each derived from previous ones. In these cases, we usually have two forms of a system P: The dag-like form P, wherein lines may be re-used arbitrarily often as hypotheses of inferences, and the tree-like form $P*$, wherein a line can be used only once.

Chapter 3

BPLK and G

In this chapter we introduce the sequent system BPLK, which is basically the propositional fragment of LK enhanced with the reasoning power of Boolean programs. We then present several lemmas about BPLK and G which show that certain classes of proofs are easy to find in the two systems, and which will simplify arguments later on.

3.1 BPLK

Definition 3.1.1 (BPLK). *The system BPLK is like the propositional system LK, but with the following changes:*

1. *In addition to sequents, a proof also includes a Boolean program which defines functions. Whenever we refer to a BPLK-proof, we shall always explicitly write it as the pair $< \pi, P >$ of the proof (sequents) and the Boolean program defining the function symbols occurring in the sequents.*

2. *Formulas in sequents are formulas in the context of Boolean programs, as defined earlier.*

3. *If the Boolean program contains a definition of the form*

$$f(\overline{p}) := A(\overline{p}),$$

the new LK rules f : **left**

$$\frac{A(\overline{\phi}), \Gamma \longrightarrow \Delta}{f(\overline{\phi}), \Gamma \longrightarrow \Delta}$$

and f : **right**

$$\frac{\Gamma \longrightarrow \Delta, A(\overline{\phi})}{\Gamma \longrightarrow \Delta, f(\overline{\phi})}$$

may be used, where $\overline{\phi}$ are precisely as many formulas as $\overline{p}$ are variables.

4. *(***Substitution Rule***) The new inference rule* **subst**

$$\frac{\Delta(q, \overline{p}) \longrightarrow \Gamma(q, \overline{p})}{\Delta(\phi, \overline{p}) \longrightarrow \Gamma(\phi, \overline{p})}$$

may be used, where all occurrences of q have been substituted for.

One point to note about the above definition is that we allow only one substitution at a time with the **subst** rule. However, we can substitute $\overline{\phi}$ for $\overline{p}$ in the sequent $S(\overline{p})$ by first performing several instances of **subst** to change the variables $\overline{p}$ to variables which do not occur in $\overline{\phi}$ or the sequent. We then substitute each of $\overline{\phi}$ into the sequent, one at a time. The size of the proof fragment is a polynomial in the size of the resulting sequent.

3.2 Basic Results on BPLK and G

Lemma 3.2.1. *There is a polynomial r such that if ϕ is any formula in the language of either BPLK or of G, then the sequent*

$$\phi \longrightarrow \phi$$

has a proof (in the appropriate system) which can be found in time $O(r(|\phi|))$, in the case of G, and time $O(r(|\phi| + |P|))$, in the case of BPLK with P the Boolean program defining the function symbols in ϕ. The length of this proof is thus bounded by the same value.

Proof. The proof is a simple induction on the structure of ϕ:

- If ϕ is a variable then the result is immediate.

- If ϕ is $\neg\psi$, then to the sequent

$$\psi \longrightarrow \psi$$

apply $\neg$: **right** and $\neg$: **left**.

- If ϕ is $\psi \vee \theta$ then apply **weakening** then $\vee$: **right** to each of the sequents

$$\psi \longrightarrow \psi$$

and

$$\theta \longrightarrow \theta,$$

then apply $\vee$: **left**.

- The case for $\wedge$ is symmetric.

- If ϕ is $Qx\psi(x)$ for some quantifier Q then to the sequent

$$\psi(a) \longrightarrow \psi(a)$$

apply both introduction rules for the quantifier, left first for universal, right first for existential.

- If ϕ is $f(\overline{\psi})$, then by a separate induction on i, derive

$$f_i(p) \longrightarrow f_i(p),$$

using **subst** and the other cases of the current lemma to derive

$$A_i(\overline{p}) \longrightarrow A_i(\overline{p}),$$

and then applying f_i-introduction. Finally, apply **subst** to $f(\overline{p}) \longrightarrow f(\overline{p})$.

We apply at most 5 inferences for each connective in ϕ (and P), and the proof can easily be computed using a recursion on the structure of ϕ, so $r(x) = x^2$ is sufficient. $\square$

Lemma 3.2.2 (Simple Manipulations). *There is a polynomial r such that if $\overline{a}$ are variables and $\overline{A}$ are formulas, and if*

$$\Gamma'(\overline{a}) \longrightarrow \Delta'(\overline{a})$$

follows from

$$\Gamma(\overline{a}) \longrightarrow \Delta(\overline{a})$$

via a proof of size k, then a proof of the sequent S':

$$\Gamma'(\overline{A}) \longrightarrow \Delta'(\overline{A})$$

from the sequent S:

$$\Gamma(\overline{A}) \longrightarrow \Delta(\overline{A})$$

can be found in time $O(r(k, |\overline{A}|))$, ($O(r(k, |\overline{A}|, |P|))$ in the case of BPLK, with P defining all relevant function symbols) and whose length is thus similarly bounded..

Proof. Simply substitute $\overline{A}$ for $\overline{a}$ in the original proof. The only case to consider is if a sequent in the original proof is an initial sequent, in which case we can apply lemma 3.2.1. The substitution expands the original proof by at most a factor of $|\overline{A}|$ and adding the subproofs obtained from lemma 3.2.1 adds a factor of at most $5|\overline{A}|^2$ so $r(k, |\overline{A}|) = k(|\overline{A}|)^3$ is sufficient. □

Lemma 3.2.3 (Build Lemma for G and BPLK). *There are polynomial-size proofs of*

$$\overline{A} = \overline{B} \longrightarrow C(\overline{A}) = C(\overline{B}),$$

for any formulas $\overline{A}$,$\overline{B}$,C, in either language/system. In the case of BPLK, the size of P, the Boolean program defining the function symbols occurring in A, B, C, is an argument to the polynomial.

Proof. The proof is a simple induction on the structure of C.

- If C is a variable, then we need to prove $\overline{A} = \overline{B} \longrightarrow A_i = B_i$ for some i, which is trivial.

- If C is $D \wedge E$, $D \vee E$ or $\neg D$, we prove $\overline{A} = \overline{B} \longrightarrow D(\overline{A}) = D(\overline{B})$ and $\overline{A} = \overline{B} \longrightarrow E(\overline{A}) = E(\overline{B})$ and perform some simple propositional manipulations to obtain the desired sequent.

- If C is $QxD(x)$ for some quantifier Q, we first prove $\overline{A} = \overline{B} \longrightarrow D(\overline{A}, p) = D(\overline{B}, p)$. Some simple manipulations yield $\overline{A} = \overline{B}, D(\overline{A}, p) \longrightarrow D(\overline{B}, p)$ and then two quantifier introductions give $\overline{A} = \overline{B}, QxD(\overline{A}, x) \longrightarrow QxD(\overline{B}, x)$. Likewise, we obtain $\overline{A} = \overline{B}, QxD(\overline{B}, x) \longrightarrow QxD(\overline{A}, x)$ and then some simple manipulations yield the result.

- If C is $f_j(D_1, ..., D_k)$ then first prove

$$\overline{A} = \overline{B} \longrightarrow \overline{D(\overline{A})} = \overline{D(\overline{B})}$$

then apply **subst** to the sequent

$$\overline{p} = \overline{q} \longrightarrow f_j(\overline{p}) = f_j(\overline{q})$$

(which is proved by a simple induction on j) to substitute $\overline{D(\overline{A})}$ for p and $\overline{D(\overline{B})}$ for $\overline{q}$ and then **cut** to obtain the result.

□

We end with a final lemma showing that substitution is like a derived rule in G:

Lemma 3.2.4. *There is a polynomial r such that a G-proof of the sequent $S(\overline{\phi})$ from the sequent $S(\overline{p})$ (both in the language of G) can be found in time $O(r(|S(\overline{\phi})|))$.*

Proof. The proof is as follows: First apply $\neg$: **right** then $\vee$: **right** to obtain an equivalent sequent with a single formula,

$$\longrightarrow F_S(\overline{p}).$$

Apply $\forall$: **right** to obtain

$$\longrightarrow \forall \overline{x} F_S(\overline{x}).$$

By lemma 3.2.1, the sequent

$$F_S(\overline{\phi}) \longrightarrow F_S(\overline{\phi})$$

has a short proof, and thus $\forall$: **left**, **weakening** and **cut** produce

$$\longrightarrow F_S(\overline{\phi}).$$

The desired sequent follows after some simple manipulations. $\square$

Chapter 4

BPLK P-Simulates G

In this chapter we show that the system BPLK polynomially simulates G. We do so by describing a procedure for translating a sequent in the language of G into a semantically equivalent one in the language of BPLK with an accompanying Boolean program. This new Boolean program defines function symbols which occur in the translated sequent, and which replace the quantifiers and bound variables from the original one. This translation is used to translate a G-proof line-by-line into the language of BPLK, and we show how to fill in the gaps to form a correct proof in the latter system. Incidentally, this technique could easily be adapted to provide an alternate proof of the PSPACE-completeness of Boolean programs, as the proof in [11] did not involve a reduction from QBFs.

Perhaps the first kind of translation one would think of would be to use Boolean programs to calculate Skolem functions, and use these to replace variables bound by existential quantifiers. Variables bound by universal quantifiers would then become free variables. However, there are many problems with this approach. First of all, this type of translation would add free variables, so in some sense it is a less faithful translation than we would like. Second, formulas in the antecedent of a sequent occur negatively in the context of the entire sequent, so we would have to translate those quantifiers differently than those in the succedent. This means that the same formula would have two different translations depending on which side of the sequent it occurred in, and this would greatly complicate simulating the **cut** rule.

Instead, the translation we adopt is very much reminiscent of the work of Hilbert and Bernays ([17] and [18]). Those authors used a different language: instead of quantifiers they used the so-called ϵ-calculus, wherein $\exists x A(x)$ would be written as $A(\epsilon_x(A(x)))$ and $\forall x A(x)$ as $A(\epsilon_x(\neg A(x)))$. The ϵ symbol $\epsilon_x(A(x))$ represents an object which will satisfy A, if such an object exists. Those authors did not ascribe any kind of functional interpretation to the ϵ symbols, but since our universe is of only two elements, it turns out that Boolean programs can easily compute the values of such symbols, and this is where we base our translation.

An alternative translation suggested by Toniann Pitassi mirrors the definition of propositional quantifiers, and would replace $\forall x A(x)$ by $A(0) \wedge A(1)$ and $\exists x A(x)$ by $A(0) \vee A(1)$. To avoid exponential increase in formula size, each of these would then be replaced by a function symbol defined by a Boolean program. Unfortunately, the principal difficulty with the translation we do adopt, namely that the names of the function symbols in translations change after introduction of quantifiers, is also present with this translation. The complexity of the arguments would thus be similar.

4.1 Special Notation

In the translation in this chapter, we shall use function symbols whose names are based on formulas of G. Because of the delicate nature of these names, we shall be especially detailed and careful about substitutions and the names of free variables. We shall therefore, in this chapter only, modify our notation:

When a formula A is displayed as $A(\overline{p})$, we shall mean that $\overline{p}$ are *all* the free variables of A. We shall never use the notation $A(q)$ to mean a substitution, but instead that A has the single free variable q. All substitutions will be denoted with a postfix operator

$[B/p]$ which means that the formula B (which may be a single variable) is substituted for p in the formula which precedes the operator. We may write several of these in a row, and the meaning is that they are performed in sequence from left to right. We may also interject free variable lists, as in the example

$$A(a,b)[B(q,r,s)/b](q,r,s,a)[C/s][D/r].$$

4.2 A Translation from the Language of G to that of BPLK

To aid in translating we introduce the following definition, which gives us a well-defined merging operator which says how to combine several Boolean programs into one, eliminating duplicate definitions:

Definition 4.2.1 (Merging $P \diamond Q$ of Boolean Programs). *If P and Q are Boolean programs, or at least fragments (not necessarily defining all function symbols used in definitions), then the merging $P \diamond Q$ of P and Q is obtained by first concatenating P and Q, and then deleting all but the first definition of each function symbol.*

We shall use this merging operator in the following translation, and later on in the actual simulation. However, whenever we merge two Boolean programs which both define a particular function symbol, it will always be the case that the two definitions are identical. Thus, which of the definitions is kept is immaterial.

Now we can present the actual translation, defined first for single formulas:

Definition 4.2.2 (Translation $< \ulcorner\phi\urcorner, P[\phi] >$ of ϕ). *We recursively define a translation of a quantified propositional* semiformula *into a semantically equivalent quantifier-free* formula *in the language of BPLK, together with a Boolean program defining the function symbols in that formula.*

- *If ϕ is an atom p then $< \ulcorner\phi\urcorner, P[\phi] >$ is $< \phi, \emptyset >$.*

- *If ϕ is $\psi \wedge \theta$ ($\psi \vee \theta$) then $< \ulcorner\phi\urcorner, P[\phi] >$ is $< \ulcorner\psi\urcorner \wedge \ulcorner\theta\urcorner, P[\psi] \diamond P[\theta] >$ ($< \ulcorner\psi\urcorner \vee \ulcorner\theta\urcorner, P[\psi] \diamond P[\theta] >$).*

- *If ϕ is $\neg\psi$ then $< \ulcorner\phi\urcorner, P[\phi] >$ is $< \neg\ulcorner\psi\urcorner, P[\psi] >$.*

- *If ϕ is $\exists x\psi(x, \overline{p})$, then $< \ulcorner\phi\urcorner, P[\phi] >$ is $< f_\phi(\overline{p}), P[\psi] \diamond P'[\phi] >$ Here, $P'[\phi]$ is a Boolean program fragment with the following two definitions:*

$$\epsilon_\phi(\overline{p}) := \ulcorner\psi\urcorner[1/x](\overline{p})$$

$$f_\phi(\overline{p}) := \ulcorner\psi\urcorner[\epsilon_\phi(\overline{p})/x](\overline{p}).$$

- *If ϕ is $\forall x\psi(x, \overline{p})$, then $< \ulcorner\phi\urcorner, P[\phi] >$ is $< f_\phi(\overline{p}), P[\psi] \diamond P'[\phi] >$ Here, $P'[\phi]$ is a Boolean program fragment with the following two definitions:*

$$\epsilon_\phi(\overline{p}) := \ulcorner\psi\urcorner[0/x](\overline{p})$$

$$f_\phi(\overline{p}) := \ulcorner\psi\urcorner[\epsilon_\phi(\overline{p})/x](\overline{p}).$$

Thus, in the case where a quantifier is the top-level connective of $\phi = Qx\psi(x, \overline{p})$, we first pick a function symbol whose name depends on the formula ϕ (i.e., including more deeply nested quantifiers and free variables). This choice encodes both which variable is the most newly quantified one, and also what kind of quantifier it is. We call this function symbol a witnessing function for ψ because its value will satisfy ψ if possible, in case Q is existential, or else it will falsify ψ if possible.

We then substitute this function symbol, applied to all the variables free in ϕ, into the *translation* of ψ. Finally, we define a new function symbol using the resulting formula, and use it instead of the formula. We do so because $\ulcorner\psi\urcorner[\epsilon_\phi(\overline{p})/x](\overline{p})$ has more occurrences of free variables than $\ulcorner\psi\urcorner(\overline{p})$, and so without the new function symbol, subsequent substitutions of this form may increase the length of the formula exponentially.

Note that if ϕ is quantifier-free, then it is unchanged by the translation and the Boolean program which results is empty.

Definition 4.2.3. *If S is the sequent*

$$\Gamma_1, ..., \Gamma_j \longrightarrow \Delta_1, ..., \Delta_k$$

then $< \ulcorner S \urcorner, P[S] >$ is

$$< \ulcorner \Gamma_1 \urcorner, ..., \ulcorner \Gamma_j \urcorner \longrightarrow \ulcorner \Delta_1 \urcorner, ..., \ulcorner \Delta_k \urcorner \qquad , \qquad P[\Gamma_1] \diamond ... \diamond P[\Delta_k] > .$$

We call $P[\phi]$ or $P[S]$ the Boolean program *arising* from the translation. As a final lemma in this section, we show that translations are polynomial size:

Lemma 4.2.4. *If S is a sequent in the language of G, then $| < \ulcorner S \urcorner, P(S) > | \in O(|S|^3)$.*

Proof. First, note that $|\ulcorner \phi \urcorner| \in O(|\phi|)$: Inductively following the recursive definition, all the Boolean connective cases increase the translation size linearly. In the case of $\phi \equiv \exists x \psi(x, \overline{p})$, the translation $\ulcorner \phi \urcorner$ is just $f_\phi(\overline{p})$, so the number of symbols in the translation is at most twice the number in the original formula. This is because to write the translation we write just an f, a copy of ϕ, and then a list of ϕ's free variables.

Now, the Boolean program arising from a translation of ϕ contains at most two function symbols for each quantifier in ϕ. The size of the names of the function symbols is linear in $|\phi|$, since the name is based on a subformula of ϕ. The defining formula for an ϵ function symbol is linear in size from the previous paragraph, since it is essentially just a translation of a subformula of ϕ. The defining formula for an f function symbol consists of a translation of a subformula of ϕ, of linear size, with an ϵ function symbol substituted for one of the variables, and therefore is at most quadratic in size. The size of the entire Boolean program is thus in $O(|S|^3)$. $\square$

4.3 A Simulation of G by BPLK

First, we show that the translations defined above are semantically equivalent to the original formulas.

Lemma 4.3.1. *In the presence of the Boolean program $P[\phi]$ defining the function symbols as above (arising from the translation), a (possibly) quantified Boolean formula ϕ is semantically equivalent to its translation $\ulcorner\phi\urcorner$.*

Proof. The proof is by a simple induction on the structure of ϕ. The interesting cases are when ϕ is $\exists x\psi(x,\overline{p})$ or $\forall x\psi(x,\overline{p})$. In the first case, if the translation holds then the value obtained by evaluating the witnessing function satisfies ψ, and so ϕ holds. Conversely, if ϕ holds then either 0 or 1 satisfies ψ, and it is easy to see that the witnessing function will evaluate to a satisfying value.

In the second case, observe that the witnessing function will falsify ψ if possible. Thus if the translation holds then ψ is satisfied by both 0 and 1. Conversely, if ϕ holds then ψ is satisfied no matter what the witnessing function evaluates to.

Note that the correctness of the cases of $\vee$ and $\wedge$ depends on the fact that whenever two Boolean programs are merged while translating a QBF, they never disagree on the definition of any function symbol. Thus, for example, the value of $\ulcorner\psi \vee \theta\urcorner$ with respect to $P[\psi \vee \theta]$ is indeed the Boolean $\vee$ of the values of $\ulcorner\psi\urcorner$ and $\ulcorner\theta\urcorner$, each with respect to their own Boolean programs. □

Now, the operations of substitution and translation do not commute directly even in the case of substituting only a single variable, so for example if $A(a) \equiv \exists x(a \vee x)$, then its translation is $\ulcorner A\urcorner(a) \equiv f_{\exists x(a\vee x)}(a)$. If we then substitute b for a we obtain $\ulcorner A\urcorner[b/a] \equiv f_{\exists x(a\vee x)}(b)$. On the other hand, if we did these things in the reverse order we would get $\ulcorner A[b/a]\urcorner \equiv f_{\exists x(b\vee x)}(b)$, which is different. A technical lemma is needed to resolve this difficulty:

Lemma 4.3.2. *Let $A(s, \overline{p}, \overline{r})$ be a semiformula and $B(\overline{p}, \overline{q})$ a formula, both in the language of G, so B is automatically free for s in A, and let $\overline{p}$ be all free variables except s which are common to A and B. (s may be in $\overline{q}$) Then proofs of*

$$\ulcorner A[B/s]\urcorner(\overline{p}, \overline{q}, \overline{r}) \longrightarrow \ulcorner A\urcorner[\ulcorner B\urcorner/s](\overline{p}, \overline{q}, \overline{r})$$

and

$$\ulcorner A\urcorner[\ulcorner B\urcorner/s](\overline{p}, \overline{q}, \overline{r}) \longrightarrow \ulcorner A[B/s]\urcorner(\overline{p}, \overline{q}, \overline{r})$$

can be found in time polynomial in the size of translations and the size of the Boolean program arising from them.

Proof. We prove the lemma by induction on the structure of A. The base case is if A is atomic. If A is s, then $\ulcorner A\urcorner$ is s, so $\ulcorner A[B/s]\urcorner \equiv \ulcorner B\urcorner \equiv \ulcorner A\urcorner[\ulcorner B\urcorner/s]$. Otherwise A is some other variable u so $\ulcorner A[B/s]\urcorner \equiv \ulcorner A\urcorner[\ulcorner B\urcorner/s] \equiv u$. In either case, apply lemma 3.2.1.

If A is not atomic then we have the following cases for each possible main connective:

- A is $C \vee D$: By assumption, we have proofs of $\ulcorner C[B/s]\urcorner \longrightarrow \ulcorner C\urcorner[\ulcorner B\urcorner/s]$ and $\ulcorner D[B/s]\urcorner \longrightarrow \ulcorner D\urcorner[\ulcorner B\urcorner/s]$. Use **weakening** and $\vee$: **right** to produce new sequents with succedent $\ulcorner C\urcorner[\ulcorner B\urcorner/s] \vee \ulcorner D\urcorner[\ulcorner B\urcorner/s] \equiv \ulcorner(C \vee D)\urcorner[\ulcorner B\urcorner/s]$, and then apply $\vee$: **left**. The converse sequent is proved similarly.

- The case for $\wedge$ is symmetric and that for $\neg$ is similarly easy.

- A is $QxC(x, s, \overline{p}, \overline{r})$: In this case, we have already found a proof π_1 of $\ulcorner C[B/s]\urcorner \longrightarrow \ulcorner C\urcorner[\ulcorner B\urcorner/s]$, and a proof π_2 of the converse sequent. We must find proofs of $\ulcorner A[B/s]\urcorner \longrightarrow \ulcorner A\urcorner[\ulcorner B\urcorner/s]$ and its converse. Now, the first step is to derive

$$\longrightarrow \ulcorner A[B/s]\urcorner = \ulcorner C[B/s]\urcorner[\epsilon_{A[B/s]}(\overline{p}, \overline{q}, \overline{r})/x] \qquad (4.3.1)$$

and

$$\longrightarrow \ulcorner A\urcorner[\ulcorner B\urcorner/s] = \ulcorner C\urcorner[\epsilon_A(s, \overline{p}, \overline{r})/x][\ulcorner B\urcorner/s] \equiv \ulcorner C\urcorner[\ulcorner B\urcorner/s][\epsilon_A(\ulcorner B\urcorner, \overline{p}, \overline{r})/x], \qquad (4.3.2)$$

which follow directly from the defining equations for $\ulcorner A[B/s]\urcorner \equiv f_{A[B/s]}(\overline{p},\overline{q},\overline{r})$ and $\ulcorner A\urcorner \equiv f_A(s,\overline{p},\overline{r})$, respectively.

The next step is to derive

$$\longrightarrow \epsilon_{A[B/s])}(\overline{p},\overline{q},\overline{r}) = \epsilon_A(\ulcorner B\urcorner,\overline{p},\overline{r}). \tag{4.3.3}$$

First, use **subst** on the endsequent of π_1 from the induction hypothesis to substitute 1 (or 0, as appropriate for Q) for x to obtain $\ulcorner C[B/s]\urcorner[1/x](\overline{p},\overline{q},\overline{r}) \longrightarrow \ulcorner C\urcorner[\ulcorner B\urcorner/s][1/x](\overline{p},\overline{q},\overline{r})$. Then, $\epsilon_{A[B/s]}$ introduction is applied on the left and ϵ_A introduction on the right. The process is repeated with π_2 and the introduction rules used on the opposite sides of the sequent, and the equality follows.

The final step is as follows: First substitute $\epsilon_{A[B/s]}(\overline{p},\overline{q},\overline{r})$ for x in the endsequent of π_1. We obtain a proof of

$$\ulcorner C[B/s]\urcorner[\epsilon_{A[B/s]}(\overline{p},\overline{q},\overline{r})/x] \longrightarrow \ulcorner C\urcorner[\ulcorner B\urcorner/s][\epsilon_{A[B/s]}(\overline{p},\overline{q},\overline{r})/x]. \tag{4.3.4}$$

Then use the equality (4.3.3) derived in the previous paragraph with the build lemma to obtain a proof of

$$\ulcorner C\urcorner[\ulcorner B\urcorner/s][\epsilon_{A[B/s]}(\overline{p},\overline{q},\overline{r})/x] \longrightarrow \ulcorner C\urcorner[\ulcorner B\urcorner/s][\epsilon_A(\ulcorner B\urcorner,\overline{p},\overline{r})/x]. \tag{4.3.5}$$

Now, using (4.3.4) and (4.3.5) we can easily obtain

$$\ulcorner C[B/s]\urcorner[\epsilon_{A[B/s]}(\overline{p},\overline{q},\overline{r})/x] \longrightarrow \ulcorner C\urcorner[\ulcorner B\urcorner/s][\epsilon_A(\ulcorner B\urcorner,\overline{p},\overline{r})/x].$$

The desired sequent $\ulcorner A[B/s]\urcorner \longrightarrow \ulcorner A\urcorner[\ulcorner B\urcorner/s]$ is then obtained using (4.3.1) and (4.3.2).

The proof of the other desired sequent is obtained symmetrically to the final step above, but starting with π_2.

By induction therefore, we can obtain π_1' and π_2', proofs of the desired sequents, in polynomial time. $\square$

The main result of the chapter follows:

Theorem 4.3.3. *If S is a quantifier-free sequent with a proof π_1 in G, then S has a BPLK-proof $< \pi_2, P[\pi_1] >$ which, given π_1, can be found in time polynomial in $|\pi_1|$ (and thus has polynomial size).*

Proof. First of all, $P[\pi_1]$ is formed by merging the Boolean programs arising from translating all the sequents in π_1. More precisely, if π_1 is $S_1, ..., S_k$, then $P[\pi_1] = P[S_1] \diamond ... \diamond P[S_k]$.

We then form π_2 directly by translating π_1 sequent-by-sequent into the language of BPLK, and when necessary adding some extra sequents between the translated ones. We have the following cases, depending on the inference rule used:

- Observe that all initial sequents of G are their own translations, and are also initial sequents of BPLK.

- If S is non-initial and is inferred by *any* rule except a quantifier introduction with hypothesis(es) T (and U), then $\ulcorner S \urcorner$ follows from $\ulcorner T \urcorner$ (and $\ulcorner U \urcorner$) by the same rule. This is because translations of identical formulas are identical, and the translation operator commutes with the connectives $\vee$, $\wedge$ and $\neg$.

- If S is inferred from T by the introduction of a universal quantifier on the right, or that of an existential quantifier on the left, then considering the first case we have that T is

$$\Gamma \longrightarrow \Delta, C(a, \overline{p})$$

and S is

$$\Gamma \longrightarrow \Delta, \forall x(C[x/a](x, \overline{p})).$$

Their translations are thus

$$\ulcorner \Gamma \urcorner \longrightarrow \ulcorner \Delta \urcorner, \ulcorner C \urcorner(a, \overline{p})$$

and

$$\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, f_{\forall x C[x/a]}(\overline{p}),$$

respectively. For convenience, let $A \equiv C[x/a]$. In this notation, $\ulcorner T\urcorner$ is

$$\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, \ulcorner A[a/x]\urcorner(a, \overline{p}).$$

First, $\ulcorner A[a/x]\urcorner \longrightarrow \ulcorner A\urcorner[\ulcorner a\urcorner/x]$ follows from lemma 4.3.2, taking $B \equiv a$, and $s \equiv x$. $\ulcorner T\urcorner$ and **cut** then yield

$$\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, \ulcorner A\urcorner[a/x].$$

Next, observe that a cannot occur in Γ or Δ (or therefore their translations) due to the restriction on $\forall$: **right**. So, when we apply **subst** to this sequent, substituting $\epsilon_{\forall x A}(\overline{p})$ for a to obtain

$$\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, \ulcorner A\urcorner[\epsilon_{\forall x A}(\overline{p})/x],$$

$\ulcorner\Gamma\urcorner$ and $\ulcorner\Delta\urcorner$ are unchanged. Finally, $\ulcorner A\urcorner[\epsilon_{\forall x A}(\overline{p})/x] \longrightarrow f_{\forall x A}(\overline{p})$ follows from the definition of the function symbol $f_{\forall x A}$ using the introduction **right** rule for that symbol. $\ulcorner S\urcorner$ follows with **weakening** and **cut**.

- The interesting case is when S follows from T by the introduction of an existential quantifier on the right, or that of a universal quantifier on the left. The two cases are symmetrical, so consider the first. T is

$$\Gamma \longrightarrow \Delta, A(x, \overline{p})[B/x]$$

and S is

$$\Gamma \longrightarrow \Delta, \exists x(A(x, \overline{p})).$$

The translations $\ulcorner T\urcorner$ and $\ulcorner S\urcorner$ are

$$\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, \ulcorner A[B/x]\urcorner$$

and

$$\ulcorner \Gamma \urcorner \longrightarrow \ulcorner \Delta \urcorner, f_{\exists x A}(\overline{p}).$$

Thus, it suffices to prove $\ulcorner A[B/x] \urcorner \longrightarrow f_{\exists x A}(\overline{p})$ and apply **cut**.

First, we derive $\ulcorner A[B/x] \urcorner \longrightarrow \ulcorner A \urcorner[\ulcorner B \urcorner/x]$ using lemma 4.3.2.

The next task is to prove

$$\ulcorner A \urcorner[\ulcorner B \urcorner/x] \longrightarrow \ulcorner A \urcorner[\epsilon_{\exists x A}(\overline{p})/x].$$

This sequent has a proof involving 4 applications of the build lemma. In this proof sketch we omit the corner brackets and the subscript on $\epsilon_{\exists x A}$. We first show that either 1 satisfies A or not. Next, if 1 satisfies A then ϵ must, since in this case ϵ will evaluate to one. We then show that if 1 does not satisfy A, but B does, then ϵ must also, since in this case B and ϵ will both evaluate to 0. Finally, we combine these three pieces to get the result.

1.	$\epsilon(\overline{p}) := A[1/x]$	*BP Def'n*
2.	$\longrightarrow A[1/x], \neg A[1/x]$	*Easy*
3.	$\epsilon(\overline{p}) \longrightarrow A[1/x]$	*1*
4.	$A[1/x] \longrightarrow \epsilon(\overline{p})$	*1*
5.	$A[1/x], 1 \longrightarrow \epsilon(\overline{p})$	*4,* **weakening**
6.	$A[1/x], \epsilon(\overline{p}) \longrightarrow 1$	$\longrightarrow 1$, **weakening**
7.	$A[1/x], A[1/x] \longrightarrow A[\epsilon(\overline{p})/x]$	*5, 6, build lemma*
8.	$A[1/x] \longrightarrow A[\epsilon(\overline{p})/x]$	*7,* **contraction**
9.	$\neg A[1/x], \epsilon(\overline{p}) \longrightarrow 0$	*3,* $\neg$: **left, weakening**
10.	$\neg A[1/x], 0 \longrightarrow \epsilon(\overline{p})$	$0 \longrightarrow$, **weakening**
11.	$\neg A[1/x], A[0/x] \longrightarrow A[\epsilon(\overline{p})/x]$	*9, 10, build lemma*
12.	$B, B \longrightarrow 1$	$\longrightarrow 1$, **weakening**
13.	$B, 1 \longrightarrow B$	$B \longrightarrow B$, **weakening**

14. $B, A[B/x] \longrightarrow A[1/x]$ *12,13,build lemma*

15. $\neg A[1/x], A[B/x], B \longrightarrow 0$ *14,*$\neg$: **left**, **weakening**

16. $\neg A[1/x], A[B/x], 0 \longrightarrow B$ $0 \longrightarrow$, **weakening**

17. $\neg A[1/x]A[B/x], A[B/x] \longrightarrow A[0/x]$ *15, 16, build lemma*

18. $\neg A[1/x], A[B/x] \longrightarrow A[\epsilon(\overline{p})/x]$ *17,* **contraction**, *11,* **weakening**, **cut**

19. $A[B/x] \longrightarrow A[\epsilon(\overline{p})/x]$ *18, 8, 2,* **weakening**, **cut**

Lastly, $\ulcorner A \urcorner [\epsilon_{\exists x A}(\overline{p})/x] \longrightarrow f_{\exists x A}(\overline{p})$ follows from the definition of $f_{\forall x A}$

Now, starting with $\ulcorner T \urcorner$ and using each of these last three sequents in turn with **weakening** and **cut**, one obtains the sequent $\ulcorner S \urcorner$, as desired.

Finally, in the case that S is quantifier-free (for example, the final sequent in a proof), then the translation of S is S, and this completes the proof of the theorem. $\square$

Chapter 5

G P-Simulates BPLK

In this chapter we define a translation of Boolean programs into formulas in the language of G, that is, using propositional quantifiers. Ultimately, we want to translate sequents in BPLK into equivalent ones in G, analogously to the previous chapter, and we shall use the translated Boolean program to obtain the values of function symbol applications. Since a line of the program can reference previous lines (in effect using a shorter Boolean program to define the current function symbol), an inductive construction is indicated, using the translations of previous lines to obtain the translation of the current one. However, if more than one copy of previous translations are ever used to translate a line, there is a possibility that the formula sizes will increase exponentially. Therefore our translation will produce one formula which can be used to simultaneously evaluate all the function symbols in the Boolean program, and one copy of this formula will be incorporated into the translation of the next line.

The method we shall use to "multiplex" the single occurrence of a translation is inspired by a trick used by Stockmeyer and Meyer in [26] to show that the problem of evaluating a given quantified Boolean formula (QBF) is PSPACE-complete. The idea is to abbreviate $\phi(x_1, y_1) \wedge \phi(x_2, y_2)$ as $\forall x, y[((x = x_1 \wedge y = y_1) \vee (x = x_2 \wedge y = y_2)) \supset \phi(x, y)]$.

This is not exactly what is needed but is quite close.

5.1 A Translation from the Language of BPLK to that of G

Since we have an infinite supply of variables, let us reserve $\overline{v}$, $\overline{t}$, $\overline{z}$, $\overline{d}$, and $\overline{u}$ as new variables not occurring in the BPLK-proof being translated. There will be a variables v_A and t_A for each formula A in the language of BPLK, and we shall replace occurrences of function symbols by these variables as part of our translation.

Definition 5.1.1 (Hat and Corner Operators). *If $A(\overline{p})$ is a formula in the language of BPLK ($\overline{p}$ are the variables free in A), then let $\widehat{A}(\overline{p}, \overline{v})$ be the result of replacing, in A, every maximal subformula $f(\overline{B})$ of A whose main connective is a function symbol by the corresponding variable $v_{f(\overline{B})}$. This new formula will have some subset of the original $\overline{p}$ and also some v's as free variables. The corner operator (e.g. $\ulcorner A \urcorner$) is defined similarly, except that corresponding t variables are used instead of v's. If either operator is applied to a set of formulas, we mean that it should be applied to each formula in the set in turn.*

Example 5.1.2 (Hat Operator). *If $\psi(\overline{p})$ is the formula*

$$f(p_1) \vee g(f(p_1 \wedge p_2)) \wedge \neg p_3,$$

then $\widehat{\psi}(\overline{p}, \overline{v})$ is

$$v_{f(p_1)} \vee v_{g(f(p_1 \wedge p_2))} \wedge \neg p_3.$$

We shall assume that a Boolean program defines functions $f_1, f_2, ...$ and that each function symbol in the Boolean program is defined as $f_i(\overline{p}) := A_i(\overline{p})$. (In other words, A_i is the defining formula of f_i). Now, $\phi_k(\overline{\overline{z}}, \overline{u})$ will be the translation of the Boolean program up to and including the definition of f_k. The meaning of $\phi_k(\overline{z_1}, ..., \overline{z_k}, u_1, ..., u_k)$ will be that this formula is satisfied if and only if $f_1(\overline{z_1}) = u_1, ...,$ and $f_k(\overline{z_k}) = u_k$.

We are now in a position to define the translation of a Boolean program more formally:

Definition 5.1.3. *Consider a Boolean program defining $f_1 ... f_k$. Define $\phi_0 := 1$. Now, say the definition of f_i includes the function symbol occurrences $f_{j_1}(\overline{B_1})$, ..., $f_{j_m}(\overline{B_m})$ in order, some of which may be nested in others. (Here $j_1, j_2, ..., j_m$ are just the indexes of the function symbols, i.e., a sequence of m integers, not necessarily distinct, each smaller than i). Then define*

$$
\begin{aligned}
&\phi_i(\overline{z_1}, ..., \overline{z_i}, u_1, ..., u_i) := \\
&\quad \exists \overline{v_{f_j(B_j(\overline{z_i}))}} [\widehat{A_i}(\overline{z_i}, \overline{v_{f_j(B_j(\overline{z_i}))}}) = u_i \wedge \\
&\qquad \forall \overline{z'_1}, ..., \overline{z'_{i-1}}, u'_1, ..., u'_{i-1} [\phi_{i-1}(\overline{\overline{z'}}, \overline{u'}) \supset [\\
&\qquad\quad \left.\begin{aligned}
&(\overline{z'_{j_1}} = \overline{\widehat{B_1}(\overline{z_i})} \supset u'_{j_1} = v_{f_{j_1}(B_1(\overline{z_i}))}) \wedge \\
&(\overline{z'_{j_2}} = \overline{\widehat{B_2}(\overline{z_i})} \supset u'_{j_2} = v_{f_{j_2}(B_2(\overline{z_i}))}) \wedge \\
&\vdots \\
&(\overline{z'_{j_m}} = \overline{\widehat{B_m}(\overline{z_i})} \supset u'_{j_m} = v_{f_{j_m}(B_m(\overline{z_i}))}) \wedge
\end{aligned}\right\} (*) \\
&\qquad\quad \left.\begin{aligned}
&(\overline{z'_1} = \overline{z_1} \supset u'_1 = u_1) \wedge \\
&\vdots \\
&(\overline{z'_{i-1}} = \overline{z_{i-1}} \supset u'_{i-1} = u_{i-1})
\end{aligned}\right\} (**) \\
&\qquad\quad] \\
&\qquad] \\
&\quad].
\end{aligned}
$$

Example 5.1.4. *Consider the following simple contrived Boolean program:*

$$f_1(p_1, p_2) := \neg p_1 \wedge p_2$$

$$f_2(p_1, p_2) := f_1(p_1, \neg p_2) \vee \neg f_1(f_1(\neg p_2, p_1), p_1)$$

Then by our terminology, $A_1(\overline{p})$ is $\neg p_1 \wedge p_2$, and $A_2(\overline{p})$ is $f_1(p_1, \neg p_2) \vee \neg f_1(f_1(\neg p_2, p_1), p_1)$. ϕ_0 is 1. Now, In the definition of f_1 there are no occurrences of function symbols, so *there are no B_j. We thus define*

$$\phi_1(z_{1,1}, z_{1,2}, u_1) := [(\neg z_{1,1} \wedge z_{1,2}) = u_1 \wedge [\phi_0 \supset [1]]].$$

ϕ_1 has a number of degenerate portions and is therefore not so interesting. In the definition of f_2, f_1 occurs 3 times (and is the only function symbol present). By our terminology we have the following:

$$\begin{aligned}
m &= 3 \\
j_1 &= 1 \\
j_2 &= 1 \\
j_3 &= 1 \\
B_{1,1}(\overline{p}) &:= p_1 \\
B_{1,2}(\overline{p}) &:= \neg p_2 \\
B_{2,1}(\overline{p}) &:= f_1(\neg p_2, p_1) \\
B_{2,2}(\overline{p}) &:= p_1 \\
B_{3,1}(\overline{p}) &:= \neg p_2 \\
B_{3,2}(\overline{p}) &:= p_1 \\
\widehat{B_{1,1}}(\overline{z_2}, \overline{v}) &:= z_{2,1} \\
\widehat{B_{1,2}}(\overline{z_2}, \overline{v}) &:= \neg z_{2,2} \\
\widehat{B_{2,1}}(\overline{z_2}, \overline{v}) &:= v_{f_1(\neg z_{2,2}, z_{2,1})} \\
\widehat{B_{2,2}}(\overline{z_2}, \overline{v}) &:= z_{2,1} \\
\widehat{B_{3,1}}(\overline{z_2}, \overline{v}) &:= \neg z_{2,2} \\
\widehat{B_{3,2}}(\overline{z_2}, \overline{v}) &:= z_{2,1} \\
\widehat{A_2}(\overline{z_2}, \overline{v}) &:= v_{f_1(z_{2,1}, \neg z_{2,2})} \vee \neg v_{f_1(f_1(\neg z_{2,2}, z_{2,1}), z_{2,1})}
\end{aligned}$$

Now, ϕ_2 is constructed from the above as indicated in the definition

$$
\begin{aligned}
&\phi_2(\overline{z_1}, \overline{z_2}, u_1, u_2) := \\
&\quad \exists v_{f_1(z_{2,1},\neg z_{2,2})}, v_{f_1(f_1(\neg z_{2,2},z_{2,1}),z_{2,1})}[(v_{f_1(z_{2,1},\neg z_{2,2})} \vee \neg v_{f_1(f_1(\neg z_{2,2},z_{2,1}),z_{2,1})}) = u_2 \wedge \\
&\qquad \forall \overline{z'_1}, u'_1[\phi_1(\overline{z'_1}, u'_1) \supset [\\
&\qquad\quad (z'_{1,1} = z_{2,1} \wedge z'_{1,2} = \neg z_{2,2} \supset u'_1 = v_{f_1(z_{2,1},\neg z_{2,2})}) \wedge \\
&\qquad\quad (z'_{1,1} = v_{f_1(\neg z_{2,2},z_{2,1})} \wedge z'_{1,2} = z_{2,1} \supset u'_1 = v_{f_1(f_1(\neg z_{2,2},z_{2,1}),z_{2,1})}) \wedge \\
&\qquad\quad (z'_{1,1} = \neg z_{2,2} \wedge z'_{1,2} = z_{2,1} \supset u'_1 = v_{f_1(\neg z_{2,2},z_{2,1})}) \wedge \\
&\qquad\quad (z'_{1,1} = z_{1,1} \wedge z'_{1,2} = z_{1,2} \supset u'_1 = u_1) \\
&\qquad\quad] \\
&\qquad] \\
&\quad].
\end{aligned}
$$

Claim 5.1.5. *For each i,*

$$\phi_i(\overline{\overline{z}}, \overline{u})$$

is semantically equivalent to

$$f_1(\overline{z_1}) = u_1 \wedge ... \wedge f_i(\overline{z_i}) = u_i.$$

Proof. First, the statement vacuously holds for $i = 0$.

Now suppose it holds for $i - 1$. If $\phi_i(\overline{\overline{z}}, \overline{u})$ holds, then there exist v's satisfying the part of ϕ_i marked (*), which ensures that they have the same values as the function symbol applications they replace, so indeed $f_i(\overline{z_i}) = u_i$. The conjuncts (**) ensure that $f_j(\overline{z_j}) = u_j, j < i$.

Conversely, if $f_1(\overline{z_1}) = u_1 \wedge ... \wedge f_i(\overline{z_i}) = u_i$ holds, then the v's satisfying (*) (which exist and are unique) must have the correct values and so $\widehat{A_i}(\overline{z_i}, \overline{v}) = u_i$. Also, (**) is clearly satisfied, and thus all of ϕ_i is. $\square$

We can now define the translation of sequents. This translation is in the context of a BPLK-proof, so the Boolean program and the rest of the sequents in the proof are already fixed. Exactly which proof a particular translation is relative to is not indicated in the notation, but it will always be clear from the context.

Definition 5.1.6 (Translation $\ulcorner S \urcorner$ of the sequent S relative to π). *Fix a BPLK-proof π and its associated Boolean program defining $f_1, ..., f_k$. Let $\overline{f_{j_i}(\overline{C_i})}$ be a list of all subformulas in π whose main connective is a function symbol. ($\overline{C_i}$ are arguments to f_{j_i}, and again j_i are simply indexes).*

Then the sequent S,

$$\Gamma \longrightarrow \Delta,$$

is translated as the sequent $\ulcorner S \urcorner$:

$$\begin{aligned}
&\phi_k(0, ..., 0, \overline{\ulcorner C_1 \urcorner}, 0, ..., 0, d_1^1, ..., d_{j_1-1}^1, t_{f_{j_1}(\overline{C_1})}, d_{j_1+1}^1, ..., d_k^1),\\
&\vdots\\
&\phi_k(0, ..., 0, \overline{\ulcorner C_m \urcorner}, 0, ..., 0, d_1^m, ..., d_{j_m-1}^m, t_{f_{j_m}(\overline{C_m})}, d_{j_m+1}^m, ..., d_k^m),\\
&\quad \ulcorner \Gamma \urcorner \longrightarrow \ulcorner \Delta \urcorner.
\end{aligned}$$

Here the $\overline{\ulcorner C_i \urcorner}$ and the corresponding t's are in the correct places to be the arguments to, and the values of, the function symbol f_{j_i}. The d are dummy variables. We could use $t_{f_i(\overline{0})}$ instead of d_i^l (since d_i^l will be constrained to the value $f_i(\overline{0})$) but it will be convenient later on that the d's are distinct. We shall call the occurrences of ϕ_k above the prefix *of the translation, and the remainder the* suffix.

Now, these translations may have free variables that the original ones did not (t's and d's). We cannot, therefore, assert semantic equivalence of the two. However, we are concerned with proving valid sequents, and we can say something nearly as good:

The idea is that if the translation of a sequent is satisfied by some assignment, then either one of the t or d variables has an incorrect value, falsifying the corresponding instance of ϕ_k, or else they all have the correct values and the remainder of the translated sequent is satisfied. In that case, the original sequent is satisfied by the same assignment.

Conversely, if the original sequent is valid, then every assignment to the translation will either falsify one of the ϕ_k's, or else all the t's will have the correct value and thus the remainder of the sequent will be satisfied. Therefore,

Claim 5.1.7. *For any sequent S from the language of BPLK, S is valid if and only $\ulcorner S \urcorner$ is.*

The final lemma in this section shows that translations are polynomial size:

Lemma 5.1.8. *Let S be a sequent from the BPLK-proof $< \pi, P >$. Then $|\ulcorner S \urcorner| \in O(|P|^2|\pi|^2)$.*

Proof. First note that for any BPLK formula ϕ, we have $|\widehat{\phi}|, |\ulcorner \phi \urcorner| \in O(|\phi|)$. These operators add a constant number of symbols for each replacement they perform, and this number is bounded by the size of the formula.

Next, consider the construction of ϕ_i from ϕ_{i-1}. The following are added:

- 2 copies of $\widehat{A_i}$
- 2 copies of $\widehat{B}$, for each B which is the argument to a function symbol in A_i (in the section (*))
- 3 occurrences of the corresponding v variables (in the section (*) and the quantifier)
- section (**) whose size is in $O(|P|)$.

Therefore summing these all up for ϕ_0 through ϕ_k we see that the last item dominates the sum and that $|\phi_k| \in O(|P|^2)$.

Finally, $\ulcorner S \urcorner$ consists of the prefix, at most $|\pi|$ occurrences of ϕ_k, each with substitutions of size at most $|\pi|$, followed by the suffix, of size $O(|S|)$. Therefore $|\ulcorner S \urcorner| \in O(|P|^2|\pi|^2)$. $\square$

5.2 A Simulation of BPLK by G

We first show that proofs of sequents from two special classes are efficient to find.

Lemma 5.2.1 (Existence Sequents). *There is a polynomial r such that for every i, the sequent E_i:*

$$\longrightarrow \forall\overline{\overline{z}}\exists\overline{u}\phi_i(\overline{\overline{z}}, \overline{u})$$

has a proof which can be found in time $O(r(|E_i|))$, and whose length is thus similarly bounded.

Lemma 5.2.2 (Uniqueness Sequents). *There is a polynomial r such that for every i, the sequent U_i:*

$$\begin{aligned}\longrightarrow \forall\overline{\overline{z_1}}, \overline{\overline{z_2}}, \overline{u_1}, \overline{u_2}[\phi_i(\overline{\overline{z_1}}, \overline{u_1}) \wedge \phi_i(\overline{\overline{z_2}}, \overline{u_2}) \supset [\\ (\overline{z_{1,1}} = \overline{z_{2,1}} \supset u_{1,1} = u_{2,1})\wedge \\ \vdots \\ (\overline{z_{1,i}} = \overline{z_{2,i}} \supset u_{1,i} = u_{2,i}) \\ ] \\ ]\end{aligned}$$

has a proof which can be found in time $O(r(|U_i|))$, and whose length is thus similarly bounded.

Proof. These two lemmas are proved by induction in parallel.

For $i = 0$, the result is trivial.

Now assume the two lemmas are proved for $i - 1$. Let $\overline{\overline{B}} = \overline{B_1}, ..., \overline{B_m}$ be all formulas appearing as arguments to function symbols in the definition of f_i, $\overline{B_w}$ as arguments to

f_{j_w}. Existence and uniqueness for ϕ_{i-1} plus some simple manipulations give

$$
\begin{array}{l}
\longrightarrow \exists \overline{v}[\forall \overline{z'_1}, ..., \overline{z'_{i-1}}, u'_1, ..., u'_{i-1}[\phi_{i-1}(\overline{\overline{z'}}, \overline{u'}) \supset [\\
\quad\quad \vdots \\
\quad\quad (\overline{z'_{j_w}} = \overline{\widehat{B_w}} \supset u'_{j_w} = v_{f_{j_w}(\overline{B_w})}) \wedge \\
\quad\quad \vdots \\
\quad\quad\quad (\overline{z'_j} = \overline{z_j} \supset u'_j = u_j) \wedge \\
\quad\quad\quad \vdots \\
\quad\quad\quad] \\
\quad\quad] \\
\quad].
\end{array}
$$

Some more simple manipulations (simply conjoining the tautology $\widehat{A_i}(\overline{z_i}, \overline{v_i}) = \widehat{A_i}(\overline{z_i}, \overline{v_i})$ inside the outermost quantifier) give

$$
\begin{array}{l}
\longrightarrow \exists \overline{v}[\widehat{A_i}(\overline{z_i}, \overline{v_i}) = \widehat{A_i}(\overline{z_i}, \overline{v_i}) \wedge \\
\quad\quad \forall \overline{z'_1}, ..., \overline{z'_{i-1}}, u'_1, ..., u'_{i-1}[\phi_{i-1}(\overline{\overline{z'}}, \overline{u'}) \supset [\\
\quad\quad\quad \vdots \\
\quad\quad\quad (\overline{z'_{j_w}} = \overline{\widehat{B_w}} \supset u'_{j_w} = v_{f_{j_w}(\overline{B_w})}) \wedge \\
\quad\quad\quad \vdots \\
\quad\quad\quad (\overline{z'_j} = \overline{z_j} \supset u'_j = u_j) \wedge \\
\quad\quad\quad \vdots \\
\quad\quad\quad] \\
\quad\quad] \\
\quad],
\end{array}
$$

and then $\exists$: **right** (on the u's and one instance of $\widehat{A_i}$) and $\forall$: **right** (on the z's) yield the existence sequent for ϕ_i.

Now in the case of uniqueness, note that

$$\phi_i(\overline{\overline{z}}, \overline{u}) \longrightarrow \phi_{i-1}(\overline{\overline{z}}, \overline{u})$$

has a short proof using existence and uniqueness for $i-1$ and some simple manipulations. Thus,

$$\begin{aligned}\longrightarrow \phi_i(\overline{\overline{z_1}}, \overline{u_1}) \wedge \phi_i(\overline{\overline{z_2}}, \overline{u_2}) \supset [\\ &(\overline{z_{1,1}} = \overline{z_{2,1}} \supset u_{1,1} = u_{2,1})\wedge \\ &\vdots \qquad\qquad\qquad\qquad (*) \\ &(\overline{z_{1,i-1}} = \overline{z_{2,i-1}} \supset u_{1,i-1} = u_{2,i-1}) \\ &]\end{aligned}$$

follows by uniqueness for $i-1$. Now we proceed as follows:

First by the definition of ϕ_i,

$$\phi_i(\overline{\overline{z_1}}, \overline{u_1}) \wedge \phi_i(\overline{\overline{z_2}}, \overline{u_2}) \longrightarrow$$
$$\exists \overline{v}[\widehat{A_i}(\overline{z_{1,i}}, \overline{v}) = u_{1,i} \wedge ...] \wedge \exists \overline{v}[\widehat{A_i}(\overline{z_{2,i}}, \overline{v}) = u_{2,i} \wedge ...].$$

Then renaming the quantified variables and doing some simple manipulations,

$$\phi_i(\overline{\overline{z_1}}, \overline{u_1}) \wedge \phi_i(\overline{\overline{z_2}}, \overline{u_2}) \longrightarrow$$
$$\exists \overline{v_1}, \overline{v_2}[[\widehat{A_i}(\overline{z_{1,i}}, \overline{v_1}) = u_{1,i} \wedge ...] \wedge [\widehat{A_i}(\overline{z_{2,i}}, \overline{v_2}) = u_{2,i} \wedge ...]].$$

Uniqueness for $i-1$ and more manipulations allows us to prove that the v's in one of the conjuncts are equal to those in the other, and thus produce

$$\phi_i(\overline{\overline{z_1}}, \overline{u_1}) \wedge \phi_i(\overline{\overline{z_2}}, \overline{u_2}) \longrightarrow$$
$$\exists \overline{v_1}, \overline{v_2}[[\widehat{A_i}(\overline{z_{1,i}}, \overline{v_1}) = u_{1,i} \wedge ...] \wedge [\widehat{A_i}(\overline{z_{2,i}}, \overline{v_1}) = u_{2,i} \wedge ...]].$$

We can similarly consolidate the z's by adding a hypothesis:

$$\phi_i(\overline{\overline{z_1}}, \overline{u_1}) \wedge \phi_i(\overline{\overline{z_2}}, \overline{u_2}) \longrightarrow$$
$$\overline{z_{1,i}} = \overline{z_{2,i}} \supset \exists \overline{v_1}, \overline{v_2}[\widehat{A_i}(\overline{z_{1,i}}, \overline{v_1}) = u_{1,i} \wedge \widehat{A_i}(\overline{z_{1,i}}, \overline{v_1}) = u_{2,i}].$$

Contracting,

$$\phi_i(\overline{\overline{z_1}}, \overline{u_1}) \wedge \phi_i(\overline{\overline{z_2}}, \overline{u_2}) \longrightarrow \overline{z_{1,i}} = \overline{z_{2,i}} \supset \exists \overline{v_1}, \overline{v_2}[u_{1,i} = u_{2,i}].$$

We can now drop the quantifier:

$$\phi_i(\overline{\overline{z_1}}, \overline{u_1}) \wedge \phi_i(\overline{\overline{z_2}}, \overline{u_2}) \longrightarrow \overline{z_{1,i}} = \overline{z_{2,i}} \supset [u_{1,i} = u_{2,i}].$$

Some simple manipulations to combine this last sequent with (*), and then $\neg$: **right** and several applications of $\forall$: **right** produce the uniqueness sequent for i. $\square$

Finally, we can state and prove the main result:

Theorem 5.2.3. *If S has a BPLK-proof π_1, then S has a G-proof π_2 which, given π_1, can be found in time polynomial in $|\pi_1|$ (and thus has polynomial size).*

Proof. We construct π_2 directly by translating π_1, sequent-by-sequent, into the language of G, relative to the Boolean program of π_1. If necessary, we insert sequents to prove the translation of a sequent from the translations of its hypotheses.

First of all, if S is an initial sequent of BPLK, then it is function symbol free and so its translation is itself, and thus already an initial sequent of G.

Now consider a non-initial sequent S inferred from previous ones. If the inference was **weakening**, **contraction**, or introduction of $\neg$, $\wedge$ or $\vee$, then the same rule yields $\ulcorner S \urcorner$.

If $S = T(\psi)$ is inferred from $T(p)$ by **subst**, then note that without loss of generality we may assume that p does not occur in ψ. Otherwise we could modify π_1 to perform **subst** twice; once to substitute $\psi(q)$ for p (q is a variable which does not occur in T) and then again to substitute p for q. To simulate the substitution in G, first use lemma 3.2.4 to substitute $\ulcorner \psi \urcorner$ for p in $\ulcorner T \urcorner(p, \overline{t})$, obtaining $\ulcorner T \urcorner(\ulcorner \psi \urcorner, \overline{t})$. Finally, apply lemma 5.2.4, which follows after this proof.

The last case in the proof is when S is inferred by f_i-introduction, introducing $f_i(\overline{B(\overline{p})})$. Then clearly the sequent

$$\phi_k(\ldots), \ldots \longrightarrow \ulcorner A_i(\overline{B}) \urcorner(\overline{p}, \overline{t}) = t_{f_i(\overline{B})},$$

together with some simple manipulations, will produce $\ulcorner S \urcorner$ (basically just by using **cut**). We derive the desired sequent as follows: First, the following is straightforward:

$$\phi_k(\overline{0}, \overline{\ulcorner B \urcorner}, \overline{u}, t_{f_i(\overline{B})}, \overline{d}) \longrightarrow \phi_i(\overline{0}, \overline{\ulcorner B \urcorner}, \overline{u}, t_{f_i(\overline{B})}, \overline{d}).$$

Next, we add the rest of the prefix:

$$\phi_k, \ldots \longrightarrow \phi_i(\overline{0}, \overline{\ulcorner B \urcorner}, \overline{u}, t_{f_i(\overline{B})}, \overline{d}).$$

Expanding the ϕ_i,

$$\phi_k, \ldots \longrightarrow \exists \overline{v}[\widehat{A_i(\overline{z_i})}(\overline{\ulcorner B \urcorner}, \overline{v}) = t_{f_i(\overline{B})} \wedge \ldots.].$$

Note that the A_i occurrence above contains t variables, from the $\overline{\ulcorner B \urcorner}$ substituted for the $\overline{z_i}$, and also v variables, from function symbols occurring in the definition of f_i. Uniqueness for ϕ_{i-1} and

$$\phi_k(\overline{0}, \overline{\ulcorner C(\overline{B}) \urcorner}, \overline{u}, t_{f_j(\overline{C(\overline{B})})}, \overline{d}) \longrightarrow \phi_{i-1}(\overline{0}, \overline{\ulcorner C(\overline{B}) \urcorner}, \overline{u}, t_{f_j(\overline{C(\overline{B})})}, \overline{d}),$$

one sequent for each function symbol occurrence $f_j(\overline{C(\overline{(}z_i))})$ in the definition of f_i, allow us to rename the $v_{f_j(\overline{C(\overline{z_i})})}$ in the occurrence of $\widehat{A_i}$ above to $t_{f_j(\overline{C(\overline{B})})}$, producing $\ulcorner A_i(\overline{B}) \urcorner$, and then we drop the existential quantifier and some conjuncts to get

$$\phi_k(\ldots), \ldots \longrightarrow \ulcorner A_i(\overline{B}) \urcorner = t_{f_i(\overline{B})},$$

which is the desired sequent.

Nearing the end of the proof now, if S is the last sequent of the proof, then it is function symbol-free. We need only remove the prefix from $\ulcorner S \urcorner$ to obtain S. The t variable corresponding to the outer-most function symbol application in π_1 (there may be many outer-most applications) is defined by an occurrence of ϕ_k, but it is not used in the definition of any of the other t variables. We may thus use $\exists$: **left** on the t and the d's, followed by $\forall$: **left** on the B's and the 0's, to change this occurrence into $\forall \overline{\overline{z}} \exists \overline{u} \phi_k(\overline{\overline{z}}, \overline{u})$, which we can cut away with the existence sequent and **weakening**. We can now do the same for the next most outer function symbol application, and so on. The resulting sequent at the end of this process is S, which completes the proof. $\square$

All that remains is to prove lemma 5.2.4. This lemma is analogous to lemma 4.3.2 of the previous chapter, and is needed because substitution does not commute with translation.

Lemma 5.2.4. *If $T(p)$ is a sequent in a BPLK-proof and ψ is a BPLK formula in which p does not occur, then a G-proof of $\ulcorner T(\psi)\urcorner$ from $\ulcorner T\urcorner(\ulcorner\psi\urcorner)$ can be found in time polynomial in the size of its endsequent.*

Proof. The first step is to use simple manipulations to rename all the variables t in $\ulcorner T\urcorner(\ulcorner\psi\urcorner)$. A variable $t_{B(p)}$ is renamed to $t_{B(\psi)}$ by an application of lemma 3.2.4. This renaming can be done in any order, and call the resulting sequent U. Now, it is easy to see that for every occurrence of a subformula of the form $\ulcorner C(p)\urcorner$ in $\ulcorner T\urcorner$, the corresponding occurrence in U is $\ulcorner C(\psi)\urcorner$: This follows because whenever the translation operator replaces a subformula $B(p)$ of $C(p)$ by a function symbol, the symbol's name is $t_{B(p)}$, and so after the renaming it will be $t_{B(\psi)}$ as it should be.

Now, consider any variable $t_{f_i(\overline{B(p)})}$ occurring in $\ulcorner T\urcorner$. This variable is defined by an occurrence of ϕ_k in the prefix of $\ulcorner T\urcorner$:

$$\phi_k(0, ..., 0, \overline{\ulcorner B(p)\urcorner}, 0, ..., 0, d_1^1, ..., d_{i-1}^1, t_{f_i(\overline{B(p)})}, d_{i+1}^1, ..., d_k^1).$$

(In fact, it is possible that this variable occurs only in the prefix.) After the substitution of $\ulcorner\psi\urcorner$ into $\ulcorner T\urcorner$, the corresponding occurrence became

$$\phi_k(0, ..., 0, \overline{\ulcorner B(\overline{p})\urcorner(\ulcorner\psi\urcorner)}, 0, ..., 0, d_1^1, ..., d_{i-1}^1, t_{f_i(\overline{B(p)})}, d_{i+1}^1, ..., d_k^1).$$

After the renaming, in U this occurrence becomes

$$\phi_k(0, ..., 0, \overline{\ulcorner B(\overline{\psi})\urcorner}, 0, ..., 0, d_1^1, ..., d_{i-1}^1, t_{f_i(\overline{B(\psi)})}, d_{i+1}^1, ..., d_k^1),$$

which correctly defines $t_{f_i(\overline{B(\psi)})}$.

Now before the final step, note that the suffix of U is identical to the suffix of $\ulcorner T(\psi)\urcorner$, and those occurrences of ϕ_k defining t variables in the suffix of $\ulcorner T(\psi)\urcorner$ also occur in U. The only difference, then, between U and $\ulcorner T(\psi)\urcorner$ is that the former sequent may have some prefix formulas which the latter does not, and vice versa. We can thus use the existence sequents (or **contraction**, in the case of a duplicate) to cut away the superfluous prefix formulas from U, and **weakening** to add the missing ones. The result is the desired sequent. $\square$

Chapter 6

Future Work and Conclusions

In this thesis we demonstrated a strong connection between two propositional proof systems both based on PSPACE reasoning. These results raise many interesting questions which remain unsolved:

6.1 A Technical Improvement

First of all, from a technical perspective it would be nice to get rid of the **subst** rule from BPLK. It is shown in Dowd [15] that extended Frege systems p-simulate substitution Frege. Boolean programs would appear to be a generalization of the extension rule, so it seems reasonable that a similar result to Dowd's might hold which would allow a version of BPLK without **subst** to p-simulate the **subst**-augmented version.

6.2 Witnessing and Search Problems

Buss and Krajíček in [5] show that those functions which are Σ_1^b definable in T_2^1 are exactly polynomial time projections of PLS functions. PLS is Papadimitriou's class of polynomial local search problems and is discussed in [19], [24] and [27]. Because of the correspondence between T_2^1 and G_1, it is therefore the case that the problem of finding witnesses for the quantifiers in a proof in G_1 is also exactly as hard as PLS.

Several lines of research are suggested: First, it would be interesting to characterize the hardness of the witnessing problems for the other subsystems of G, and indeed different kinds of definability in the subsystems of T_2 and S_2. Part of this work has recently been done by Chiari and Krajíček in [8] for Σ_2^b and Σ_3^b definability in T_2^2 but nothing general is known yet. Secondly, there are other local search problems than PLS, some of which are discussed in [19] and in more detail in [2]. It would be interesting to find propositional proof systems whose witnessing problems were exactly projections of these other local search problem classes.

6.3 Subsystems of BPLK

Another set of questions which are particularly interesting concerns the possibility of finding natural subsystems of BPLK, akin to the structure of G. In their paper [11], the authors find a natural restriction of Boolean programs, essentially amounting to extension axioms, for witnessing proofs in G_1^*. It would be instructive to find restrictions of Boolean programs which would naturally witness proofs in other subsystems of G. It would also be interesting to find some kind of a hierarchy within BPLK which may or may not correspond to the hierarchy in G.

6.4 Miscellaneous

Finally, it is possible that due to the apparent ease of use of BPLK, more positive results may be forthcoming than with G. For example, it may not be too difficult to produce polynomial-sized proofs in BPLK of some of the conjectured hard examples for Frege [3] and extended Frege systems. As another example, the connection between G and U_2^1, which currently is restricted to only Σ_1^b formulas, might be generalized to handle more general theorems, in particular including the second-order features of that system.

Bibliography

[1] M. Ajtai. The complexity of the pigeonhole principle. In *29th Annual Symposium on Foundations of Computer Science*, pages 346–355, White Plains, New York, 24–26 October 1988. IEEE.

[2] Paul Beame, Stephen Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1):3–19, August 1998.

[3] Maria Luisa Bonet, Samuel R. Buss, and Toniann Pitassi. Are there hard examples for frege systems? In *P. Clote, J. Remmel (eds.): Feasible Mathematics II*, pages 30–56. Birkhäuser, Boston, 1995.

[4] S. Buss. *Bounded Arithmetic.* Bibliopolis, Naples, 1986.

[5] Samuel Buss and Jan Krajíček. An application of Boolean complexity to separation problems in bounded arithmetic. *Proceedings of the London Mathematical Society*, 69:1–21, 1994.

[6] Samuel R. Buss. Relating the bounded arithmetic and polynomial time hierarchies. *Annals of Pure and Applied Logic*, 75(1–2):67–77, 12 September 1995.

[7] Samuel R. Buss, editor. *Handbook of Proof Theory.* Elsevier Science B. V., Amsterdam, 1998.

[8] Mario Chiari and Jan Krajíček. Witnessing functions in bounded arithmetic and search problems. *The Journal of Symbolic Logic*, 63(3):1095–1115, September 1998.

[9] S. A. Cook. CSC 2429S: Proof Complexity and Bounded Arithmetic. Course notes, URL: "http://www.cs.toronto.edu/~sacook/csc2429_98", Spring 1998.

[10] Stephen Cook and Robert Reckhow. On the lengths of proofs in the propositional calculus (preliminary version). In *Conference Record of Sixth Annual ACM Symposium on Theory of Computing*, pages 135–148, Seattle, Washington, 30 April–2 May 1974.

[11] Stephen Cook and Michael Soltys. Boolean programs and quantified propositional proof systems. *Bulletin of the Section of Logic*, 28(3), 1999.

[12] Stephen A. Cook. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 3–5 1971 1971.

[13] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Conference Record of Seventh Annual ACM Symposium on Theory of Computing*, pages 83–97, Albuquerque, New Mexico, 5–7 May 1975.

[14] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

[15] Martin Dowd. Model theoretic aspects of P $\neq$ NP. Typewritten manuscript, 1985.

[16] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2–3):297–308, August 1985.

[17] D. Hilbert and P. Bernays. *Grundlagen der Mathematik I*. Springer, Berlin, 1934.

[18] D. Hilbert and P. Bernays. *Grundlagen der Mathematik II*. Springer, Berlin, 1939.

[19] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, August 1988.

[20] Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory.* Cambridge University Press, 1995.

[21] Jan Krajíček and Pavel Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic*, 54(3):1063–1079, 1989.

[22] Jan Krajíček, Pavel Pudlák, and Gaisi Takeuti. Bounded arithmetic and the polynomial hierarchy. *Annals of Pure and Applied Logic*, 52(1–2):143–153, 1991.

[23] Jan Krajíček and Gaisi Takeuti. On bounded Σ_1^1 polynomial induction. In S. R. Buss and P. J. Scott, editors, *FEASMATH: Feasible Mathematics: A Mathematical Sciences Institute Workshop*, pages 259–80. Birkhauser, 1990.

[24] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, June 1994.

[25] Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, August 1997.

[26] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Conference Record of Fifth Annual ACM Symposium on Theory of Computing*, pages 1–9, Austin, Texas, 30 April–2 May 1973.

[27] Mihalis Yannakakis. Computational complexity. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55. John Wiley and Sons, Chichester, UK, 1997.

[28] D. Zambella. Notes on polynomially bounded arithmetic. *The Journal of Symbolic Logic*, 61(3):942–966, 1996.