Lemma Logging: Translating Pseudo-Boolean proofs with lemmas to VeriPB proofs

Tim Sehested Poulsen
tpw705@alumni.ku.dk

August 22, 2024

Supervisors: Jakob Nordström & Rui Ji

Abstract

Combinatorial problems have a wide range of applications, including operations research, chip design and many scheduling problems. Therefore, it is vital that the algorithms used to solve these problems produce correct results. One way to ensure correct results is the use of proof logging, where combinatorial solvers are required to produce a proof that their output is correct. This thesis takes its starting point in the proof logging tool VeriPB, which is one of the most efficient and successful in verifying proofs from combinatorial solvers. Despite its success, there are still some techniques which are currently infeasible to be verified by VeriPB. These techniques require the option to reuse other proofs as "lemmas" many times, without having to verify the proof each time it is applied. Such at type of reasoning is not supported by VeriPB. These "proofs with lemmas" can be translated into proofs valid for VeriPB, but they are quite often too long to be practical. In an attempt to deal with this issue, this thesis explores how proofs with "lemmas", can be translated more efficiently into proofs that can be verified by the VeriPB proof system. We show a way to do so, but also argue that it still might be too technical and inefficient to be practical.

Contents

1 Introduction

1.1 Combinatorial Problems

Solving combinatorial problems is fundamental to many scheduling and resource management algorithms which are used every day. With its use case ranging from; scheduling of trains, sports tournaments and kidney transplants [MTW97, YQLG16, MO15], as well as portfolio optimization in finance, genomic sequencing in biology and designing of computer chips [GB23, Wat95, Ach09], it should be no surprise that extensive research has been done to develop efficient algorithms also called *combinatorial solvers* for these types of problems.

There is a great variety of combinatorial problems, as it includes *decision problems* where the task is to answer "Yes" or "No" to whether a solution exists, as well as *optimization problems* where the goal is not only to find a solution but also to find the best one, given some criteria of optimality, and *counting problems* where the goal could be to count the number of solutions.

These problems are essential to scheduling and logistics, because all combinatorial problems, to some extent, revolves around the issue of arranging or counting objects to fit some constraints. For example, it could be that we wanted to arrange a sports tournament. When doing so, there are some real world constraints, that we must obey, such as the fact that all teams must play each other, and teams should not have to play twice in a row. The problem is then to arrange matches such that this is possible, and we might even want to minimize the number of playing fields needed at the same time. The problem of arranging the sports tournament would then have to be reformulated as a formal combinatorial problem, where a combinatorial solver could then solve it, and the result could be interpreted to see the schedule of the tournament. In computer science and mathematics, we only consider the formal problems, as they generalize a wide range of real world problems.

We have yet to describe the main characteristics of combinatorial problems, in the formal setting. One of the main characteristics of a combinatorial problem is that the collection of possible solutions, that one wishes to search through, must be *discrete*. That is each solution is distinct and separated from one another. One example could be that the solution must be an integer $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, which there are infinitely many of, but it is nevertheless still discrete. And in the scheduling of a tournament, the solution consist of determining which matches each team plays in, and for an algorithm to say that a team would play partly in one match and partly in another at the same time, would be nonsense.

Now let us look at a particular combinatorial problem. It is probably the most canonical combinatorial problem, and is known as the *Boolean Satisfiability Problem* or SAT in short. The problem is best explained through an example. In SAT we are given a logical statement such as

$$(x \vee y) \wedge (\neg x \vee \neg y) \tag{1.1}$$

where x and y are the variables and can only take the values TRUE (corresponding to 1) or FALSE (corresponding to 0). The goal is now to assign x and y a value of either 0 or 1, such that the statement above evaluates to TRUE. This statement consists of two parts, firstly the part $(x \vee y)$ which states that either x or y must be set to 1 and secondly $(\neg x \vee \neg y)$ which means that either x or y must be set to 0. One possible solution satisfying both of these criteria, could be that we set $x = 1$ and $y = 0$. SAT is thus a decision problem. Many combinatorial problems are also closely related, as the setup might be the same, but the goal ever so slightly different. For example, we could have the setup as in SAT, but instead ask "How many solutions are there?", which is known as the problem *model counting* or #SAT.

Another classical combinatorial problem, and the main one in question in this thesis, is *Pseudo-Boolean Satisfaction* also known as 0-1 integer programming. In this problem, a collection of linear inequalities over some variables is given. The objective is then to assign variables a value of 0 or 1, to satisfy all inequalities. For example, the problem could be:

$$x + y \geq 1 \tag{1.2}$$

$$-x - y \geq -1 \tag{1.3}$$

where one solution is $x = 1$ and $y = 0$. And in fact, the logical formula in equation (1.1) and these two inequalities encode exactly the same problem. Which highlights a key aspect of combinatorial problems. Often we can translate from one problem to another, so developing methods to solve one problem, could also be used to solve the another one, letting us choose whatever viewpoint is most convenient for the real world problem that we might try to model.

Many combinatorial problems have a continuous counterpart, which feel very similar but when it comes to solving them, they differ drastically. Take for example the two linear inequalities above, but instead let x and y be any real numbers, then there are infinitely many solutions to the problem, but also infinitely many wrong solutions, whereas we only had 4 possible solutions before. At first glance one might think that the problem becomes more difficult if you expand the number of possible solutions, as there are way more to sort through. But in fact, it is generally the opposite. When you allow the solution set to be continuous you gain the benefit of being able to do small adjustments and tweaks to narrow in on a solution. Whereas for discrete problems, when you take a decision on a variable, this will often force other decisions, and you can't just "tweak" the initial decision later on if any errors occur. Generally for discrete problems, you can't avoid a type of "trial and error" approach, whereas for their continuous counterpart, you can often fix mistakes done earlier without having to backtrack, which means you are always working towards a solution.

Due to the discrete nature of combinatorial problems, they are theoretically very difficult to solve. The instances we wish to solve in real life often involve thousands of variables and possibly millions of constraints, and according to the most supported hypothesis in computer science, this makes them completely unsolvable in full generality. All algorithms which have proofs that guarantee their correctness on all possible problem instances, require an astronomical amount of time to solve real world instances. This issue has led researchers to concurrently develop algorithms for which they cannot guarantee their performance, in hopes that they would be more efficient in practice. Both of these approaches were far from solving large instances until around the year 2000, where a heuristic based algorithm prevailed. This algorithm was the *Conflict Driven Clause Learning* (CDCL) algorithm [MSS99], and despite the lack of a theoretical proof of good performance, it has been used to solve large real world instances of SAT ever since. It has been at the core of all competitive SAT algorithms for the past 20 years, and the heuristic has been revised, altered, tweaked an improved upon ever since, and today it is de facto the standard for solving SAT instances, and thereby many combinatorial problems.

The lack of theoretical guarantees, and instead showcasing performance empirically, has also brought with it significant increase in the complexity of the algorithms. Algorithms nowadays combine many techniques, which makes it very difficult to theoretically reason about why the algorithms should perform well. Moreover, the algorithms are also much more difficult to implement, and it has been shown repeatedly, that the most thoroughly tested software still contain bugs [CKSW13, AGJ+18, BLB10]. This is a big issue in some cases, as it is vital that the output of the software can be trusted.

1.2 Proof Logging

The most successful solution addressing the issue of correctness, is certifying the output of an algorithm [MMNS11]. That is an algorithm is required not only to solve the problem, but also to output a *proof* or *certificate* of its correctness, such that this proof can be verified to ensure that solution is indeed correct/optimal. The research field of how these proofs can be created is *Proof Logging*. The idea is then, that an algorithm, independent of the solver, can verify the proof, such that bugs and errors can be caught automatically. The next natural thought is then; "if the first algorithm has a chance of being buggy, won't the verification algorithm as well?". To answer this there are two parts. First off, because the proof checking is performed independently of the combinatorial solver, the likelihood that both have the same bug/error is lower. But most importantly is that certificates are simple as contrasted to the combinatorial solvers, which makes it easier to reason about and implement correctly.

Certificates consist of a sequence of steps, in which each step applies a simple rule, that is easy to verify. Then at the very end of the certificate, should be some formal statement from which it follows that the solver has indeed solved the problem. The rules of reasoning that is allowed should be mathematically proven to produce correct conclusions. It is exactly these rules of reasoning or inference that makes up what we call a *proof system*. If a proof system is too complex, or its rules are inefficient to verify, then it will not be useful in practice. A secondary goal of keeping the proof system simple, is also that these verification algorithms can be *formally verified* [CFHH$^+$17, CFMSSK17], in which a specific piece of software is proven to be correct on all possible inputs, thus giving the software the highest level of assurance of correctness.

The field of SAT solving, has actively been using proof logging for a while now and there exists multiple formats for this [GN03, WHHJ14, CFHH$^+$17, CFMSSK17]. It has been used specifically for the annual SAT competition, in which algorithms compete in solving SAT instances efficiently, and since 2013, it has been mandatory for algorithms to output a proof of correctness. Specifically for SAT solving which is a decision problem, a proof of correctness when a solution exists, is quite easy, as a solution can just be outputted. Any solutions can be verified quickly, as the values can just be checked to satisfy all the constraints. The problem arises when the instance is unsatisfiable, as proving this requires more sophisticated reasoning. We generally call a proof of unsatisfiability a *refutation*. There are also many optimization problems related to SAT, which require refutations in order to prove optimality.

As time progresses more advanced methods are developed, and some of these are not supported by existing proof formats. Partly to combat this, the tool VeriPB was developed [BGMN23, GN21, Goc22], and in recent years it has shown great success by supporting proof logging for most advanced techniques used in state-of-the-art solvers, writing efficient proofs that are quick to verify. VeriPB is based on the *cutting planes* proof system [CCT87], which means it reasons on Pseudo-Boolean formulas, i.e. 0-1 linear inequalities. However, as many combinatorial problems can be translated into Pseudo-Boolean formulas, it has also shown a wide range of applicability to other combinatorial problems [GMN21, GMM$^+$20, EGM$^+$20].

Most advanced techniques used by state-of-the-art solvers are supported by VeriPB, however one that still remains unsupported is the technique *symmetric learning* [DBB17]. To support this technique the proof system needs the ability to write a proof once, and the reuse the conclusion multiple times in other proofs. The most natural parallel, is the use of lemmas in mathematics, where a lemma can be proven, and then used multiple times to prove a main theorem. It is precisely reusage of "lemmas" in proof logging that this thesis will focus on, or specifically how proofs with

lemmas can be translated into proofs that can be verified by VeriPB.

As we will see later, symmetric learning is not fundamentally unsupported by VeriPB, but at the current time, it is very inefficient to write proofs and thereby verify them for this technique. A proof system which supports symmetric learning has been proposed [TD20], but it does not support many of the other modern techniques covered by VeriPB.

1.3 Content of the thesis

In this thesis we will at first cover the fundamentals of Pseudo-Boolean reasoning, the cutting planes proof system and some aspects of VeriPB. To ease readability we will introduce quite a few bits of notation, but we will showcase that it lends itself to very natural conclusions. Once we have set the stage we will introduce what we formally will mean by a proof using lemmas in the context of Pseudo-Boolean reasoning. The final chapters then cover how such proof with lemmas, can be translated into a proof that can be verified by VeriPB. This will be done in parts, to first introduce the core ideas, and then gradually generalize them to work in all cases. At last, we will highlight that though such a translation is possible, it has its limitations.

2 Prerequisites

2.1 Pseudo-Boolean Reasoning

To formally define the problem of pseudo-boolean reasoning, we need to introduce some notation and a few other definitions first. Most of these will be the standards used in the literature, but some notation is slightly different to ease readability later on.

Definition 2.1 (Variable sets & literals)**.** Let $\vec{x} := \{x_1, x_2, \ldots, x_n\}$ be a set of variables, in which each variable x_i take value in $\{0, 1\}$. For each variable in $\vec{x}$ we also define its *negation* $x_i^0 = \overline{x_i} := 1 - x_i$, and double negation will just be the original variable, i.e. $\overline{(\overline{x_i})} = x_i$. A literal over the variable x_i will be denoted as $x_i^{\sigma_i}$, and it is either just the variable or its negation:

$$x_i^{\sigma_i} = \begin{cases} x_i & \text{if } \sigma_i = 1 \\ 1 - x_i & \text{if } \sigma_i = 0 \end{cases} \tag{2.1}$$

and the negation of a literal is quite naturally written as $x_i^{1-\sigma_i}$. The set of literals over $\vec{x}$ will be denoted as $\text{Lit}(\vec{x}) := \{x_i^{\sigma_i} \mid \forall x_i \in \vec{x},\ \forall \sigma_i \in \{0, 1\}\}$.

It is also quite common in the literature to use the standard first order logic notation of $\neg x_i$ for a negated variable and an arbitrary literal over the variable x_i is denoted as l_i. As we will be working with multiple variable sets later on, we will need to distinguish between literals over different variable sets, which the notation $x_i^{\sigma_i}$ does.

Definition 2.2 (Assignments and substitutions)**.** Let $\vec{x}$ be a variable set, a function $\rho : \vec{x} \to \vec{x} \cup \{0, 1\}$ we call an *assignment* of $\vec{x}$ if it holds that $\rho(x_i) \in \{0, 1, x_i\}$ for all x_i. The most common notation for such a function will be $\rho = \{x_1 \to b_1, x_2 \to b_2, \ldots, x_n \to b_n\}$, for $b_i \in \{0, 1, x_i\}$. If it is the case that all $b_i \in \{0, 1\}$, we call ρ a *complete assignment*, and otherwise we call it a *partial assignment* on $\vec{x}$. For partial assignments it is often the case that we only explicitly write the mapping of those x_i which are mapped to either 0 or 1, while the rest are implicitly assumed to be mapped to themselves.

If $\vec{y}$ and $\vec{x}$ are two variable sets, then an mapping $\omega : \vec{y} \to \text{Lit}(\vec{x}) \cup \{0, 1\}$ is called a *substitution* of $\vec{y}$ into $\vec{x}$, where we say the substitution can map literals as well, as follows: $\omega(y_i^{\sigma_i}) = \omega(y_i)^{\sigma_i}$ for all $y_i \in \vec{y}$. We will write substitutions as we do for assignments, and only write those y_i which are not mapped to themselves, in case $\vec{y} \subseteq \vec{x}$.

From the definition we see that for example the identity on a variable set, i.e. the function $Id : \vec{x} \to \vec{x}$ in which $Id(x_i) = x_i$ for all $x_i \in \vec{x}$, is both a partial assignment on $\vec{x}$, and a substitution of $\vec{x}$ onto itself (as long as the codomain is chosen accordingly). Now getting to the building blocks of pseudo-boolean problems, we will define the constraints.

Definition 2.3 (PB Constraints)**.** Let $\vec{x}$ be a variable set, a *constraint* C, sometimes written as $C(\vec{x})$ to indicate the variable set, is a linear inequality over literals of $\vec{x}$ on the form

$$C \doteq \left(\sum_{x_i \in \vec{x}} a_i x_i^{\sigma_i} \geq A \right) \tag{2.2}$$

for $A, a_i \in \mathbb{N}_0 := \{0, 1, 2, \ldots\}$. The values a_i we call the coefficients of C, and we will say that x_i occurs in C if $a_i \neq 0$, and likewise that x_i does not occur in C if $a_i = 0$. To distinguish between

the inequality inside the constraints and the syntactic equality, that two things are just different syntax for the same thing, we will use the symbol $\doteq$ to exactly define this.
The constant A is known as the *degree* of the constraint, and is often denoted as $\deg(C)$. Since $a_i \geq 0$, we say that a constraint is *trivial* if $A = 0$. We will define the weight of the constraint as $W(C) := \sum_{x_i \in \vec{x}} a_i$. We will often omit the indexing in the sum, and instead write $C \doteq \sum a_i x_i^{\sigma_i} \geq A$, unless the indexing is non-trivial.

If we have an assignment/substitution ω on $\vec{x}$, we can *restrict* C under ω by substituting all x_i with $\omega(x_i)$. This is written as

$$C\restriction_\omega \doteq \left(\sum a_i \omega(x_i)^{\sigma_i} \geq A\right) \tag{2.3}$$

where we notice that if $\omega(x_i) = 1$ then $\omega(x_i)^1 = 1$, $\omega(x_i)^0 = (\omega(x_i) - 1) = 0$ and if $\omega(x_i) = 0$ then quite opposite $\omega(x_i)^1 = 0$ and $\omega(x_i)^0 = 1$. Under the restriction we let anything that become a constant in the sum to be subtracted from both sides of the inequality, and if the subtraction should make the degree less than 0 we just let it be 0, and thus $C\restriction_\omega$ is again a constraint, as the degree had to be non-negative in our definition.

Here we must note that in what we have just introduced we only consider constraints in what is known as their *normalized form* in standard literature, which is when all coefficients are non-negative integers, each variable occurs only as one literal, and the inequality is "$\geq$". In some literature, one might see constraints need only to satisfy that a_i and A are integers, and we allow both "$\geq$" and "$\leq$" inequalities.

But as it is the case that any linear equality using $\leq$ and/or negative coefficients can be rewritten into normalized form [ES06], it will ease readability to only consider these.

We also say that a constraint is *satisfied* under an assignment/substitution ω if $C\restriction_\omega$ is trivial, i.e. that $\deg(C\restriction_\omega) = 0$. Similarly we say that ω *falsifies* C if it is the case that $\sum_{\omega(x_i)^{\sigma_i} \neq 0} a_i < A$, and in this case we would have that the weight of $C\restriction_\omega$, i.e. $W(C\restriction_\omega)$ is less than A. Any constraint C in which $W(C) < A$ we say that it is *unsatisfiable*, as even the assignment which makes all literals $x_i^{\sigma_i}$ in C equal 1, will leave the constraint falsified.

Definition 2.4 (PB Formula)**.** A *Pseudo-Boolean formula* is a conjunction $F \doteq \bigwedge_{j \leq m} C_j$ of constraints. It is often easier to view $F \doteq \{C_j \mid \forall j \leq m\}$ as a set of constraints, where the size of the formula $|F| = m$ is the number of constraints. The restriction of F under an assignment/substitution ω is defined as

$$F\restriction_\omega \doteq \{C_j\restriction_\omega \mid \forall j \leq m\} \tag{2.4}$$

A *solution* to F is a complete assignment ρ, in which each constraint C is satisfied by ρ.

2.2 The Cutting Planes Proof System

To reason about pseudo-boolean formulas, we will have rules of reasoning, which allow us to derive new constraints from previously known ones. The set of rules make up the proof system considered, and we will mostly be working with the cutting planes proof system [CCT87]. One of the rules in cutting planes is:

$$\textit{Literal Axiom}\ \frac{}{x_i^{\sigma_i} \geq 0} \tag{2.5a}$$

Where the notation here, means that the top parts indicate what you already know, and the bottom part is the *conclusion*, i.e. what can be derived from the top parts. In this case, that means that literal axioms have no prerequisites to be derived, and can therefore always be derived. Which makes logically sense, as literals in PB constraints only take values in $\{0, 1\}$, thus $x_i^{\sigma_i} \geq 0$ should always hold. The other rules of inference are the following:

$$\textit{Addition}\ \frac{\sum a_i x_i^{\sigma_i} \geq A \qquad \sum b_i x_i^{\delta_i} \geq B}{\sum a_i x_i^{\sigma_i} + \sum b_i x_i^{\delta_i} \geq A + B} \tag{2.5b}$$

$$\textit{Multiplication}\ \frac{\sum a_i x_i^{\sigma_i} \geq A}{\sum c a_i x_i^{\sigma_i} \geq cA}\ \ c \in \mathbb{N} \tag{2.5c}$$

$$\textit{Division}\ \frac{\sum c a_i x_i^{\sigma_i} \geq A}{\sum a_i x_i^{\sigma_i} \geq \lceil A/c \rceil}\ \ c \in \mathbb{N} \tag{2.5d}$$

For (2.5b) we will note that *cancelling addition* can occur, which is what happens when x_i and $\overline{x_i}$ are added together, and we have $x_i + \overline{x_i} = 1$. So anytime two constraints are added, we will always cancel out literals when possible and rewrite the constraint to be in normalized form. Take for example the addition of $2x_1 + 3x_2 \geq 4$ and $\overline{x_1} + 2x_3 \geq 2$ which will result in $x_1 + 3x_2 + 2x_3 \geq 5$. For the division rule we see the requirement that the divider c must divide each coefficient in the constraint, however using a combination of (2.5a) and (2.5d) we can simulate a *generalized division* rule:

$$\textit{Generalized Division}\ \frac{\sum a_i x_i^{\sigma_i} \geq A}{\sum \lceil a_i/c \rceil x_i^{\sigma_i} \geq \lceil A/c \rceil}\ \ c \in \mathbb{N} \tag{2.5e}$$

For our purposes we will swap out this rule of inference for the division rule, which will not change the proof system fundamentally [BN21].

Though the original cutting planes proof system consists only of rules (2.5a) - (2.5d), the system has been studied with a multitude of different rules extending it. As we will be working in the context of VeriPB [Goc22] we will focus only on the extension implemented in this framework, and specifically only those which are of relevance to this work. One of those rules is the *Saturation Rule*, which in some combinatorial solvers, such as sat4j [LBP10], is used instead of the division rule:

$$\textit{Saturation}\ \frac{\sum a_i x_i^{\sigma_i} \geq A}{\sum \min(A, a_i) x_i^{\sigma_i} \geq A} \tag{2.5f}$$

Here it is very important that the constraint is in normalized form, as the non-negativity of the coefficients is crucial for the rule to be defined like this. Logically it also makes sense to say that, if $a_i > A$, then it might as well be that $a_i = A$, as both have that if $x_i^{\sigma_i}$ is set to 1 in an assignment, then the constraint is satisfied.

For the remainder of the thesis we will refer to the cutting planes proof system, which will consist exactly of the rules (2.5a) - (2.5f) excluding the division rule in (2.5d). This is to some a slight deviation from the standard, but it is the system allowed in VeriPB.

2.3 Implications and Derivations

For a formula F, we say that F *models* or *implies* the constraint C if any satisfying assignment ρ to F also satisfies C. In this case we will write $F \vDash C$. Likewise, for another formula F', we would say that F implies or models F' if all constraints in F' are implied by F and write

$$F \vDash F' \iff \forall C \in F' : \ F \vDash C \tag{2.6}$$

Quite similarly if a constraint C can be derived using the cutting planes rules of inference from a formula F, i.e. using the rules (2.5a) - (2.5f), we say the C is derivable from F and write $F \vdash C$. It is not that difficult to prove that the cutting planes proof system preserve solutions [CCT87], that is, if ρ is a solution to F, and $F \vdash C$, then ρ is also a solution to C. Thus if $F \vdash C$ then we also have that $F \vDash C$. If we are able to derive the constraint $\bot \doteq 1 \geq 0$ from a formula F, then we call that derivation a *proof of unsatisfiability* of F or a *refutation* of F, because $\bot$ is impossible to satisfy, thus F is impossible to satisfy.

Generally when we talk about derivations, we not only want to know whether a cutting planes derivation exists, but rather we want to know what the derivation is. Using the notation

$$\pi : F \vdash C \tag{2.7}$$

we say that π is the derivation of C from F. Specifically $\pi = (E_1, E_2, \ldots, E_{L-1}, E_L \doteq C)$ is a sequence of constraints E_i, where each E_i is either a constraint in F, or it follows by one of the cutting planes inference rules (2.5a) - (2.5f) from constraints previous in π. Often we will also write out the constraints in our formula F, to indicate what the *premises* of the derivation are, for example:

$$\pi : P_1, P_2, \ldots, P_m \vdash C \tag{2.8}$$

instead of $\pi : F \vdash C$, when $F \doteq \{P_i \mid \forall i \leq m\}$. The constraint P_j is referred to as a *premise* of the derivation π. We will mostly view derivations as proofs for adding new constraints to our formula, and thus we would often write that if $F \vdash C$ then we can add C to F getting the formula $F' := F \cup \{C\}$, which has the same set of solutions as F. Once we start thinking of adding constraints to our original formula F, we will generally use the term *constraint database* and the symbol $\mathfrak{D}$, to emphasize that this is a set which changes over time, and preserve that F is the original formula, we are trying to refute.

Lets take a look at the following example which highlights an important derivation which we will reuse later on.

Example 2.5. A quite common constraint to observe is constraints in *disjunctive form*, i.e. constraints in which just one literal out of a set of literals must be true. So for example

$$C \doteq (x_1 + x_2 + \overline{x_3} \geq 1) \tag{2.9}$$

Often we will also look at the negation of such a literal which in this case corresponds to the constraint

$$\neg C \doteq (\overline{x_1} + \overline{x_2} + x_3 \geq 3) \tag{2.10}$$

By adding the literal axioms $x_2 \geq 0$ and $\overline{x_3} \geq 0$ to this, we thus see we get $\overline{x_1} \geq 1$. Formally we have derivation $\pi : \{\neg C\} \vdash (\overline{x_1} \geq 1)$, where

$$\pi = (\neg C, x_2 \geq 0, \overline{x_1} + \overline{x_3} \geq 2, x_3 \geq 0, \overline{x_1} \geq 1) \tag{2.11}$$

And we thus could add $\overline{x_1} \geq 1$ to our constraint database $\mathfrak{D}$ if it contained $\neg C$. We could have done similarly for x_2 and x_3 and gotten $\overline{x_2} \geq 1$ and $x_3 \geq 1$.

This example we can generalize to the following:

Proposition 2.6. *Let $C \doteq (\sum a_i x_i^{\sigma_i} \geq 1)$ be a constraint in disjunctive form, that is all $a_i \in \{0, 1\}$ then for any literal $x_i^{\sigma_i}$ occurring in C, we have that*

$$\neg C \vdash \left(x_i^{1-\sigma_i} \geq 1\right)$$

From a constraint database $\mathfrak{D}$ which contains $\neg C$, this will let us transition to $\mathfrak{D}' := \mathfrak{D} \cup \left\{x_i^{1-\sigma_i} \geq 1 \;\middle|\; \forall x_i^{\sigma_i} \in C\right\}$ This derivation also highlights a more common type of derivation which is known as a *literal implication derivation*. We say that a derivation is a literal implication derivation whenever the derivation consists only of adding literal axioms to a constraint. Another common example of using literal implication derivations, is when we wish to derive a trivial constraint, that is when the degree of the constraint is 0.

Proposition 2.7. *Let $C \doteq (\sum a_i x_i^{\sigma_i} \geq 0)$ be a trivial constraint, then $F \vdash C$, for any formula F.*

Proof. For each literal $x_i^{\sigma_i}$ in C, we simply take the literal axiom $x_i^{\sigma_i} \geq 0$ and multiply with a_i and add all of these together achieving

$$C \doteq \sum a_i x_i^{\sigma_i} \geq 0 \tag{2.12}$$

and we have derived the constraint by literal implication. □

2.4 Redundance-Based Strengthening & Reification Variables

Lastly we will introduce a rule of inference which looks quite different from the others, and is mostly used for introducing new variables which represent some constraint. Say we have a formula F over the variable set $\vec{x}$, and let $C := (\sum a_i x_i^{\sigma_i} \geq A)$ be an arbitrary constraint, which could occur in F but need not to. Say we wish to represent this constraint using only a *fresh variable* y, i.e. a variable which does not occur in F, logically speaking we would want to introduce the constraints

$$y \implies C \doteq \left(A\overline{y} + \sum a_i x_i^{\sigma_i} \geq A\right) \tag{2.13}$$

$$y \impliedby C \doteq \left((W(C) - A + 1)y + \sum a_i x_i^{1-\sigma_i} \geq W(C) - A + 1\right) \tag{2.14}$$

Where the notation $y \implies C$ and $y \impliedby C$ is just notational sugar to ease readability. But at the same time the notational sugar is really indicating how these constraints are to be thought of. Take for example (2.13), if $y = 1$ under any assignment, then to satisfy (2.13) we must satisfy C, and if $y = 0$ we don't care whether C is satisfied or not, which corresponds to the truth table for implication propositional logic. It works likewise for (2.14), if an assignment satisfy C, then to satisfy (2.14) then y must be 1.

Generally when y does not occur in F, then regardless if F is a satisfiable formula or not, we can always set y such that both (2.13) and (2.14) are satisfied. That means when equation (2.13) and (2.14) are added to F it will not change the solution set, as we for each solution can set y accordingly. When we wish to refer to both of these constraints, we do so quite naturally by writing $y \iff C$ and call y a *reification variable* of C. Reification variables are not allowed by the

standard cutting planes rules. Simply put; there is no way to introduce a new variable. To allow reification, another rule must be added to the proof system. Such a rule is commonly called the *extension rule*, and some proof systems have been studied along with such a rule. In VeriPB there is instead a slightly more general rule of inference called the *Redundance-Based Strengthening* rule, to allow reification variables, which we will introduce shortly. To understand how this rule works, we first need to understand the concept of a constraint being *redundant*.

Definition 2.8 (Redundance)**.** We say that a constraint C is *redundant* with respect to a formula F if F and $F \cup \{C\}$ are equisatisfiable.

Whenever we are trying to solve a decision problem, and specifically when we are trying to refute a formula F, it should be clear that adding redundant constraints, should not cause any problems. For the case of proof logging, we need a verifiable way to deem a constraint to be redundant, which the next proposition will help us provide. This was proved in [GN21, Proposition 3.1] and is restated here:

Proposition 2.9 (Substitution Redundancy)**.** *Let F be a formula over variable set $\vec{x}$, and $C(\vec{y})$ a constraint over variable set $\vec{y}$ which is possibly disjoint from $\vec{x}$. Then $C(\vec{y})$ is redundant with respect to F if and only if there exists a substitution $\omega : \vec{x} \cup \vec{y} \to \mathrm{Lit}(\vec{x} \cup \vec{y}) \cup \{0, 1\}$ called a witness, such that*

$$F \cup \{\neg C\} \models (F \cup \{C\}) \restriction_\omega \tag{2.15}$$

Proof. "$\implies$"

Suppose that C is redundant with respect to F. Notice that if F is unsatisfiable, then $F \vDash B$ would trivially hold for any constraint B. Thus, if F is unsatisfiable so is $F \cup \{\neg C\}$ and any substitution ω satisfies equation (2.15). If instead F and $F \cup \{C\}$ are satisfiable, we can choose ω to be a satisfying assignment to $F \cup \{C\}$. Notice now that $(F \cup \{C\}) \restriction_\omega$ is satisfied and by proposition 2.7 all these constraints can therefore be derived by the cutting planes rules from any formula, specifically also $F \cup \{\neg C\}$. As we have also argued earlier, when we have that $F \cup \{\neg C\} \vdash (F \cup \{C\}) \restriction_\omega$, we therefore also know that solutions are preserved, thus $F \cup \{\neg C\} \vDash (F \cup \{C\}) \restriction_\omega$.

"$\impliedby$"

Now suppose that we have a witness ω satisfying (2.15), and we thus needs to prove equisatisfiability. If F is unsatisfiable, then clearly so must $F \cup \{C\}$ be, as it at least need to satisfy F. In the other case, if F is satisfiable, let α be a complete satisfying assignment to F. Now define a complete assignment to $F \cup \{C\}$ as follows:

$$\alpha^*(y) := \begin{cases} 1 & \text{if } y \notin \vec{x} \\ \alpha(y) & \text{otherwise} \end{cases} \tag{2.16}$$

If α^* is a satisfying assignment to $F \cup \{C\}$, then clearly F and $F \cup \{C\}$ are equisatisfiable. If instead α^* does not satisfy $F \cup \{C\}$ we must have that α^* does not satisfy C, as F is satisfied by α. Therefore, we know that α^* satisfies $F \cup \{\neg C\}$ and by the assumption that (2.15) holds, α^* must be a complete satisfying assignment to $(F \cup \{C\}) \restriction_\omega$. Lastly define $\beta := \alpha^* \circ \omega$, and notice that $(F \cup \{C\}) \restriction_\beta = (F \cup \{C\}) \restriction_\omega \restriction_{\alpha^*}$, and thus β is a satisfying assignment to $F \cup \{C\}$, since α^* is to $(F \cup \{C\}) \restriction_\omega$, and we conclude that $F \cup \{C\}$ is also satisfiable. In conclusion F and $F \cup \{C\}$ are equisatisfiable. □

The proof really highlights that the important part is to handle solutions to F, which do not satisfy C, and ensure that they can still be mapped to another solution, thus ensuring equisatisfiability. When it comes to the problem of proof logging, simply stating the witness ω is not good enough, as the implication could be difficult to verify. Instead, we will allow a rule of inference, where the witness ω have to be specified, along with a cutting planes derivation of $(F \cup \{C\})\restriction_\omega$ from $F \cup \{\neg C\}$. Furthermore, the length of this derivation has to be polynomial in the size of F and C, otherwise it is not an efficient proof anymore. We'll state the rule cleanly here:

Rule 2.10 (Redundance-Based Strengthening)**.** A constraint C can be added to a constraint database $\mathfrak{D}$ if there exists a witness ω from variables in $\mathfrak{D}$ and C to literals over these variables or $\{0, 1\}$, along with a cutting planes derivation

$$\mathfrak{D} \cup \{\neg C\} \vdash (\mathfrak{D} \cup \{C\})\restriction_\omega$$

where the length of the derivation is at most polynomial in the size of $\mathfrak{D}$ and C.

Generally, when using rule 2.10 we need only to show that constraints in $\mathfrak{D}$ which are affected by the witness ω can be derived, as well as the constraint C which we are attempting to add. For the remainder of the thesis, we will refer to the cutting planes proof systems along with redundance-based strengthening as the *VeriPB proof system.* In reality VeriPB does have a few additional rules, but these will be of no importance to this project, therefore they are omitted here. Let us return to the example of introducing reification variables by equation (2.13) and (2.14), and let us do so by the redundance-based strengthening rule.

Proposition 2.11. *Let $\mathfrak{D}$ be a constraint database over variable set $\vec{x}$ and $C \doteq \sum a_i y_i^{\sigma_i} \geq A$ an arbitrary constraint over $\vec{y}$, which is not necessarily disjoint from $\vec{x}$. Let also z be a fresh variable, that is $z \notin \vec{x}$, and $z \notin \vec{y}$. Then we can add the constraints $z \iff C$ to $\mathfrak{D}$ using the redundance-based strengthening rule.*

Proof. Let us start by adding the constraint $z \impliedby C$ to $\mathfrak{D}$. We need to find a witness, such that the derivation in (2.15) holds. To do so we use the witness $\omega_1 = \{z \to 1\}$. Because z is fresh, this witness has no effect on $\mathfrak{D}$ and the implication $\mathfrak{D} \vDash \mathfrak{D}\restriction_{\omega_1}$ holds trivially. The constraint $(z \impliedby C)\restriction_{\omega_1}$ is satisfied, thus by proposition 2.7 we can derive it from any formula, specifically from $\mathfrak{D} \cup \{\neg (z \impliedby C)\}$.

To introduce the constraint $z \implies C$ to the formula $\mathfrak{D}' := \mathfrak{D} \cup \{(z \impliedby C\})$ we need to do a bit more work as z is not fresh anymore. In this case we will use the witness $\omega_0 = \{z \to 0\}$. By the same argument as before $(z \implies C)\restriction_{\omega_0}$ is satisfied and can thus be derived easily. Thus, we need to derive $\mathfrak{D}'\restriction_{\omega_0}$ from $\mathfrak{D}' \cup \{\neg (z \implies C)\}$, in which deriving $\mathfrak{D}\restriction_{\omega_0}$ from $\mathfrak{D}$ is yet again trivial as they are equal. Lastly we must derive $(z \impliedby C)\restriction_{\omega_0}$, and we will do so from the constraint $\neg(z \implies C)$, as we observe that

$$\neg(z \implies C) \doteq A \cdot z + \sum a_i x_i^{1-\sigma_i} \geq W(C) + 1 \tag{2.17}$$

in normalized form, by using the rewrite that $x_i^{\sigma_i} = 1 - x^{1-\sigma_i}$. From this we multiply A to the literal axiom $\overline{z} \geq 0$ achieving $A\overline{z} \geq 0$, this is then added to the constraint (2.17) to get the desired constraint

$$(z \impliedby C)\restriction_{\omega_0} \doteq \sum a_i x_i^{1-\sigma_i} \geq -A + W(C) + 1 \tag{2.18}$$

Which means we have that

$$\mathfrak{D}' \cup \{\neg (z \implies C)\} \vdash (\mathfrak{D} \cup \{z \iff C\}) \restriction_{\omega_0} \tag{2.19}$$

and thus $z \iff C$ can be added to $\mathfrak{D}$.

□

Usually when we introduce reification variables, we will use the notation $r(C)$ instead of z to denote the variable which is the reification of C.

$$r(C) \iff C = \begin{cases} A\overline{r(C)} + \sum a_i x_i^{\sigma_i} \geq A \\ (W(C) - A + 1) r(C) + \sum a_i x_i^{1-\sigma_i} \geq W(C) - A + 1 \end{cases} \tag{2.20}$$

In the following section we will showcase through different propositions, the abstraction that reification variables allow us to make, and specifically how our syntax can be used to abstract away some details. Moreover, it also serves the goal of introducing statements which will be of use to us later in the thesis, and lets the reader get accommodated with both the syntax and the types of arguments we will be making.

2.5 Syntax Abstraction

Firstly we will introduce a generalization of some syntax which we already have, namely we will for a variable set $\vec{z}$ and a constraint $C(\vec{x}) \doteq \sum a_i x_i^{\sigma_i} \geq A$ use the notation:

$$\left(\bigwedge_{z_i \in \vec{z}} z_i^{\delta_i} \implies C(\vec{x}) \right) \doteq \left(A \sum_{z_i \in \vec{z}} z_i^{1-\delta_i} + \sum a_i x_i^{\sigma_i} \geq A \right) \tag{2.21}$$

and sometimes we will instead write

$$\left(\bigwedge_{z_i \in \vec{z}} z_i^{\delta_i} \implies \left(\sum a_i x_i^{\sigma_i} \geq A \right) \right) \tag{2.22}$$

when we want to showcase what the constraint $C(\vec{x})$ is.

This syntax captures the same intuition as a logical implication, exactly that if just one literal $z_i^{\delta_i}$ is false under an assignment then the whole of (2.21) is satisfied, but if all literals are true under an assignment, then to satisfy (2.21) we must satisfy $C(\vec{x})$. Really digesting and understanding this syntax will be very important later in the thesis, as it is a powerful abstraction, and will allow us to reason much easier about many constraints. Moreover, it lends itself to very natural conclusions requiring only a few steps, as we will see in the following propositions. This first proposition, confirms that reification variables really are interchangeable with the constraints which they represent, specifically in our syntactic implications as introduced above. This will allow us to interchange between $r(C) \geq 1$ and C as desired.

Proposition 2.12 (Reification Interchangeability)**.** *Let $\vec{x}$ and $\vec{z}$ be two disjoint variable sets. Let $C \doteq \sum a_i x_i^{\sigma_i} \geq A$ be an arbitrary constraint that we have reified, and let $r(C)$ be the reification variable of C. Then we have the derivation*

$$\left\{ \bigwedge_{z_i \in \vec{z}} z_i^{\delta_i} \implies C \right\} \cup \{r(C) \impliedby C\} \vdash \left(\bigwedge_{z_i \in \vec{z}} z_i^{\delta_i} \implies (r(C) \geq 1) \right)$$

and oppositely we also have the derivation

$$\left\{\bigwedge_{z_i\in\vec{z}} z_i^{\delta_i} \implies (r(C)\geq 1)\right\}\cup\{r(C) \implies C\}\vdash\left(\bigwedge_{z_i\in\vec{z}} z_i^{\delta_i} \implies C\right)$$

both requiring 2 steps.

Proof. The derivations here are quite simple, as they are mostly a matter of adding the constraints. First recall that $W(C)\doteq\sum_i a_i$ and the constraints are:

$$\left(\bigwedge_{z_i\in\vec{z}} z_i^{\delta_i} \implies C\right)\doteq\left(A\sum z_i^{1-\delta_i}+\sum a_i x_i^{\sigma_i}\geq A\right) \tag{2.23}$$

$$\left(\bigwedge_{z_i\in\vec{z}} z_i^{\delta_i} \implies (r(C)\geq 1)\right)\doteq\left(\sum z_i^{1-\delta_i}+r(C)\geq 1\right) \tag{2.24}$$

$$(r(C) \impliedby C)\doteq\left((W(C)-A+1)\cdot r(C)+\sum a_i x_i^{1-\sigma_i}\geq W(C)-A+1\right) \tag{2.25}$$

$$(r(C) \implies C)\doteq\left(A\cdot\overline{r(C)}+\sum a_i x_i^{\sigma_i}\geq A\right) \tag{2.26}$$

From addition of equation (2.23) and (2.25) we get

$$A\sum z_i^{1-\delta_i}+\sum a_i x_i^{\sigma_i}+(W(C)-A+1)\cdot r(C)+\sum a_i x_i^{1-\sigma_i}\geq A+W(C)-A+1 \tag{2.27}$$

$$\doteq A\sum z_i^{1-\delta_i}+(W(C)-A+1)\cdot r(C)+W(C)\geq W(C)+1 \tag{2.28}$$

$$\doteq A\sum z_i^{1-\delta_i}+(W(C)-A+1)\cdot r(C)\geq 1 \tag{2.29}$$

Performing saturation here leaves us with the desired constraint

$$\sum z_i^{1-\delta_i}+r(C)\geq 1\doteq\left(\bigwedge_{z_i\in\vec{z}} z_i^{\delta_i} \implies (r(C)\geq 1)\right) \tag{2.30}$$

Similarly, if we multiply equation (2.24) with A and add it to (2.26) we get

$$A\sum z_i^{1-\delta_i}+A\cdot r(C)+A\cdot\overline{r(C)}+\sum a_i x_i^{\sigma_i}\geq A+A \tag{2.31}$$

$$\doteq A\sum z_i^{1-\delta_i}+\sum a_i x_i^{\sigma_i}\geq A \tag{2.32}$$

which is the desired constraint $\bigwedge_{z_i\in\vec{z}} z_i^{\delta_i} \implies C$.

□

Notice that we can thus also interchange between $r(C)\geq 1$ and C in any formula if one of them has been derived, which is the case if we in the proposition have $\vec{z}\doteq\emptyset$. The next proposition is also quite natural. It allows us to remove literals from an implication, if we already know the literals must be 1.

Proposition 2.13 (Implication Removal)**.** *Let $\vec{z}$ and $\vec{x}$ be disjoint variable set, and $C \doteq \sum a_i x_i^{\sigma_i} \geq A$ an arbitrary constraint. For a fixed $z_j \in \vec{z}$ we have the following cutting planes derivation*

$$\left\{ \bigwedge_{z_i \in \vec{z}} z_i^{\delta_i} \implies C \right\} \cup \left\{ z_j^{\delta_j} \geq 1 \right\} \vdash \bigwedge_{z_i \in \vec{z} \setminus z_j} z_i^{\delta_i} \implies C$$

requiring only 2 steps.

Proof. It is merely a calculation. Recall that we have

$$\left(\bigwedge_{z_i \in \vec{z}} z_i^{\delta_i} \implies C \right) \doteq \left(A \sum_{z_i \in \vec{z}} z_i^{1-\delta_i} + \sum a_i x_i^{\sigma_i} \geq A \right) \tag{2.33}$$

and we can multiply $z_j^{\delta_j} \geq 1$ with A, thus getting $A z_j^{\delta_j} \geq A$ and adding this to (2.33) we will get that $A z_j^{1-\delta_j}$ will be cancelled out from the sum, and we will get

$$A z_j^{\delta_j} + A z_j^{1-\delta_j} + A \sum_{z_i \in \vec{z} \setminus z_j} z_i^{1-\delta_i} + \sum a_i q_i^{\sigma_i} \geq A + A \tag{2.34}$$

$$\doteq \left(A \sum_{z_i \in \vec{z} \setminus z_j} z_i^{\delta_i} + \sum a_i q_i^{\sigma_i} \geq A \right) \doteq \left(\bigwedge_{z_i \in \vec{z} \setminus z_j} z_i^{\delta_i} \implies C \right) \tag{2.35}$$

as desired.

□

The next proposition shows how our syntax allows us to make derivations, which mimic transitivity in propositional logic[1], such as $p \implies q$ and $q \implies w$ lets us conclude $p \implies w$.

Proposition 2.14 (Implication transitivity)**.** *Let $\vec{x}$ and $\vec{z}$ be two disjoint variable sets, and let w be a variable not occurring in either. There exists a cutting planes derivation*

$$\left\{ \bigwedge_{z_i \in \vec{z}} z_i \implies (x_j \geq 1) \;\middle|\; \forall x_j \in \vec{x} \right\} \cup \left\{ \bigwedge_{x_j \in \vec{x}} x_j \implies (w \geq 1) \right\} \vdash \bigwedge_{z_i \in \vec{z}} z_i \implies (w \geq 1)$$

of length $|\vec{x}| + 1$.

Proof. Recalling the constraints

$$\bigwedge_{z_i \in \vec{z}} z_i \implies (x_j \geq 1) \doteq \left(\sum_{z_i \in \vec{z}} \overline{z_i} + x_j \geq 1 \right) \tag{2.36}$$

$$\bigwedge_{x_j \in \vec{x}} x_j \implies (w \geq 1) \doteq \left(\sum_{x_j \in \vec{x}} \overline{x_j} + w \geq 1 \right) \tag{2.37}$$

[1] In propositional logic it is called hypothetical syllogism

From adding (2.36) for all $x_j \in \vec{x}$ together and denoting $|\vec{x}| = n$, we get the following

$$n \cdot \sum_{z_i \in \vec{z}} \overline{z_i} + \sum_{x_j \in \vec{x}} x_j \geq n \tag{2.38}$$

if we add this to the constraint (2.37) we get the following

$$n \cdot \sum_{z_i \in \vec{z}} \overline{z_i} + \sum_{x_j \in \vec{x}} x_j + \sum_{x_j \in \vec{x}} \overline{x_j} + w \geq n + 1 \tag{2.39}$$

$$\doteq n \cdot \sum_{z_i \in \vec{z}} \overline{z_i} + w \geq 1 \tag{2.40}$$

because x_j and $\overline{x_j}$ cancel out. When (2.40) is saturated we get the desired constraint $\bigwedge_{z_i \in \vec{z}} z_i \implies (w \geq 1)$. □

It could be generalized more, allowing all z_i and x_j to instead be literals, but as we will only need this proposition in cases where we know the variable isn't negated, we will keep it to this more simple form.

The following lemma that we will prove, is both the most cumbersome one to prove, but also quite essential for this thesis. It will allow us to take any standard cutting planes derivation, $\pi : P_1, P_2, \ldots, P_m \vdash C$ and instead run it on reified premises $r\left(P_j\right) \implies P_j$, in which the conclusion is instead a single constraint capturing the whole derivation, and looks like $\bigwedge_{j=1}^{m} r\left(P_j\right) \implies C$. This will be a very important tool later on, as it will allow us to "perform" any cutting planes derivation inside another derivation, as long as we are able to introducing reification variables for the premises first.

Lemma 2.15 (Axiom reification)**.** *Let $\pi : P_1, P_2, \ldots, P_m \vdash C$ be a cutting planes derivation over variables $\vec{x}$. Using instead the axioms $r\left(P_j\right) \implies P_j$ for all $j \leq m$, we can derive the constraint $\bigwedge_{j=1}^{m} r\left(P_j\right) \implies C$, that is there exists a cutting planes derivation*

$$\pi' : r\left(P_1\right) \implies P_1,\ \ldots,\ r\left(P_m\right) \implies P_m \vdash \bigwedge_{j=1}^{m} r\left(P_j\right) \implies C$$

over variables $\vec{x} \cup \left\{r(P_j) \mid j \leq m\right\}$, and with length $L' = O(m \cdot L)$, where L is the length of the original derivation π.

Proof. Here we will view derivations as a sequence of constraints, and specifically we will write the derived constraints in π as $\pi = (E_1, E_2, \ldots, E_L)$, and to accommodate the difference in derivation length between π and π', we will use subindexing in π', to denote steps which does not correspond to any step in π. Thus, we will denote the steps in π' as

$$\pi' = (E'_{0.1}, E'_{0.2}, \ldots, E'_1, E'_{1.1}, E'_{1.2}, \ldots, E'_2, E'_{2.1}, \ldots, E'_L) \tag{2.41}$$

We will then proof by induction that for each $s \in \mathbb{N}$ and $s \leq L$, the constraint E'_s in the derivation π' is on the form $\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_s$. From this it then follows that since $E_L \doteq C$ we have that $E'_L \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies C\right)$.

The base case $s = 1$

This is quite trivial as there are only two options for what E_1 can be:

- If E_1 is a premise P_j for some $j \leq m$, we take the corresponding premise $r\left(P_j\right) \implies P_j$ in π', and add the literal axiom $r\left(P_{j'}\right) \geq 0$ for all $j' \leq m$, but $j' \neq j$. This will give use the desired constraint, and we let $(E'_{0.1}, E'_{0.2}, \ldots, E'_1)$ be this derivation to get $E'_1 \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies P_j\right)$.
- If E_1 is a literal axiom P_j, we do the exact same thing and take the literal axiom $x_i^{\sigma_i} \geq 0$ and add the literal axiom $r\left(P_{j'}\right) \geq 0$ for all $j' \leq m$. This gives us the derivation $(E'_{0.1}, E'_{0.2}, \ldots, E'_1)$ where $E'_1 \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies (x_i^{\sigma_i} \geq 0)\right)$.

Induction step $s \geq 2$.

By our induction hypothesis we know that $(E'_1, E'_{1.1}, \ldots, E'_2, \ldots, E'_{s-1})$ is a valid derivation in which $\forall k < s$ and $k \in \mathbb{N}$, we have that

$$E'_k \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_k\right) \tag{2.42}$$

We have to show that E'_s is also on this form, regardless of what the derivation step is in π. We have 6 options for the derivation step:

- *Case 1 (Axiom) $E_s \doteq P_j$ for some $j \leq m$:*

  This works equivalent to the argument in the base case, and we are able to derive $E'_s \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies P_j\right)$
- *Case 2 (Literal Axiom) $E_s \doteq (x_i^{\sigma_i} \geq 0)$:*

  Again this is equivalent to the argument in the base case, and we can derive $E'_s \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies (x_i^{\sigma_i} \geq 0)\right)$
- *Case 3 (Division) $E_s \doteq E_k/c$ for $k < s$ and $c \in \mathbb{N}$:*

  By the induction hypothesis we know $E'_k \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_k\right)$. Let us denote $E_k \doteq (\sum a_i x_i^{\sigma_i} \geq A)$. Through inspection of E'_k and E'_k/c we see that

$$E'_k \doteq A \sum_{j=1}^{m} \overline{r\left(P_j\right)} + \sum a_i x_i^{\sigma_i} \geq A \tag{2.43}$$

$$E'_k/c \doteq \lceil A/c \rceil \sum_{j=1}^{m} \overline{r\left(P_j\right)} + \sum \lceil a_i/c \rceil x_i^{\sigma_i} \geq \lceil A/c \rceil \tag{2.44}$$

$$\tag{2.45}$$

  Which by definition is the same as $\bigwedge_{j=1}^{m} r\left(P_j\right) \implies (E_k/c)$, therefore we let

$$E'_s \doteq E'_k/c \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_k/c\right) \tag{2.46}$$

and thus E'_s is on the correct form, and required only 1 step.

- *Case 4 (Addition)* $E_s \doteq E_{k_1} + E_{k_2}$ *for* $k_1, k_2 < s$.
  By our induction hypothesis we know that

$$E'_{k_1} \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_{k_1} \right) \tag{2.47}$$

$$E'_{k_2} \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_{k_2} \right) \tag{2.48}$$

  and let us denote $E_{k_1} \doteq (\sum a_i x_i^{\sigma_i} \geq A)$ and $E_{k_2} \doteq \left(\sum b_i x_i^{\delta_i} \geq B\right)$. From addition of E'_{k_1} and E'_{k_2} we get

$$E'_{k_1} + E'_{k_2} \doteq A \sum_{j=1}^{m} \overline{r\left(P_j\right)} + B \sum_{j=1}^{m} \overline{r\left(P_j\right)} + \sum a_i x_i^{\sigma_i} + \sum b_i x_i^{\delta_i} \geq A + B \tag{2.49}$$

$$\doteq (A+B) \sum_{j=1}^{m} \overline{r\left(P_j\right)} + \sum a_i x_i^{\sigma_i} + \sum b_i x_i^{\delta_i} \geq A + B \tag{2.50}$$

  This is essentially the desired form, but we have to account for cancellations happening across the two sums $\sum a_i x_i^{\sigma_i} + \sum b_i x_i^{\delta_i}$. If some cancellations happens, the degree of the constraint decrease, and then the coefficient $(A+B)$ of $\overline{r\left(P_j\right)}$ are larger than the degree. If we perform saturation we ensure that the coefficient of $\overline{r\left(P_j\right)}$ is exactly the degree of the constraint, and thus after saturating equation (2.50) we get the desired constraint

$$E'_s \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies (E_{k_1} + E_{k_2}) \right) \tag{2.51}$$

- *Case 5 (Multiplication)* $E_s \doteq c \cdot E_k$ *for* $k < s$ *and* $c \in \mathbb{N}$*:*
  Again by the induction hypothesis we know that $E'_k \doteq \bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_k$, and we again denote $E_k \doteq (\sum a_i x_i^{\sigma_i} \geq A)$. We quickly see that we can let $E'_s \doteq c \cdot E'_k$ since

$$c \cdot E'_k \doteq c \cdot \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_k \right) \tag{2.52}$$

$$\doteq (c \cdot A) \sum_{j=1}^{m} \overline{r\left(P_j\right)} + \cdot \sum c a_i x_i^{\sigma_i} \geq c \cdot A \tag{2.53}$$

$$\doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies c \cdot E_k \right) \tag{2.54}$$

  and then E'_s is as desired.

- *Case 6 (Saturation) E_s is the saturation of E_k for a $k < s$:*
  Let $E_k \doteq (\sum a_i x_i^{\sigma_i} \geq A)$. By the induction hypothesis

$$E'_k \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_k \right) \doteq \left(A \sum_{j=1}^{m} \overline{r\left(P_j\right)} + \sum a_i x_i^{\sigma_i} \geq A \right) \tag{2.55}$$

  which when saturated gives us exactly $\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_s$, thus we just saturate E'_k to get $E'_s \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_s \right)$.

Thus, we finally conclude, that despite which cutting planes rule of inference that is performed in π to get step E_s, we can achieve a subderivation in π' such that $E'_s \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_s \right)$.

This lets us conclude that

$$E'_L \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies E_L \right) \doteq \left(\bigwedge_{j=1}^{m} r\left(P_j\right) \implies C \right) \tag{2.56}$$

As we have seen in the different arguments, despite which rule of inference is used, the equivalent subderivation in π' is most often just one or two steps, despite when we have to deal with literal axioms and premises from F, in which case we require $O(m)$ steps. This means that in the worst case the derivation π' will have length $O(m \cdot L)$, but for most realistic derivations, the length will actually be within a constant factor of L.

□

We will also need a more general form of the lemma just shown, in which we allow any "condition" on our premises. That is instead of replacing the premise P_j with $r\left(P_j\right) \implies P_j$, we will instead associate a set of literals V_j with each premise P_j, and then use $\bigwedge_{l_{j,i} \in V_j} l_{j,i} \implies P_j$ instead. Notice as we can choose $V_j \doteq \emptyset$, we can choose to alter only some of the premises. This generalized form would directly prove lemma 2.15 by choosing $V_j \doteq \left\{ r\left(P_j\right) \right\}$, but since the proofs are basically equivalent, we have chosen only to include the proof of lemma 2.15, which for the uninitiated is much easier to comprehend. We will still state the generalized form without proof, as we will need it later in the thesis.

Proposition 2.16 (Conditional derivation)**.** *Let $\pi : P_1, P_2, \ldots, P_m \vdash C$ be a cutting planes derivation over variables $\vec{x}$. Let $\mathcal{U}$ be a set of literals disjoint from* $\mathrm{Lit}(\vec{x})$*, and for each $j \leq m$ let $V_j \subseteq \mathcal{U}$ be a subset associated with P_j. Using instead the premises*

$$\bigwedge_{l_{j,i} \in V_j} l_{j,i} \implies P_j$$

we can derive the constraint $\bigwedge_{l \in \mathcal{U}} l \implies C$, that is there exists a cutting planes derivation

$$\pi' : \bigwedge_{l_{1,i} \in V_1} l_{1,i} \implies P_1, \bigwedge_{l_{2,i} \in V_2} l_{2,i} \implies P_2, \ldots, \bigwedge_{l_{m,i} \in V_m} l_{m,i} \implies P_m, \vdash \bigwedge_{l \in \mathcal{U}} l \implies C$$

with length $L' = O(|\mathcal{U}| \cdot L)$, where L is the length of the original derivation π.

3 Proofs with lemmas

In this chapter we will discuss what it will mean to have "lemmas" in a proof, or rather what it will mean to reuse derivations without having to carry them out. Allowing derivations to be reused, won't introduce fundamentally new ways of reasoning. If the derivation was valid before, it would also be so even if we were to do it on different variables. This actually holds generally for any substitution of variables that if $\pi : P_1, \dots, P_m \vdash D$ is a valid cutting planes derivation, then we could also perform a derivation under a substitution λ resulting in a cutting planes derivation $\pi\restriction_\lambda : P_1\restriction_\lambda, \dots, P_m\restriction_\lambda \vdash D\restriction_\lambda$. The formal arguments for why it generally holds that we can perform a derivation under a substitution, are very similar to those used to prove lemma 2.15, and will not be given here. But it also makes sense, that if we have already carried out the derivation for π, we shouldn't have to perform it again, and instead just be able to reuse the conclusion of π to conclude $\pi\restriction_\lambda$ in a single step.

In this chapter we will try to formalize this type of reasoning, and what a potential rule allowing this could look like. One practical issue when defining the structure of a proof with lemma, is that during execution of an algorithm it might be difficult to tell which subderivation that should be viewed as a lemma. It is not until the algorithm wishes to achieve a similar conclusion that it is known which parts actually makes up the lemma. Just as in mathematics you rarely know your lemmas before the main theorem. Formally speaking, say π is the main derivation in question, and we wish to apply a lemma in step i in π. Let $\pi^* : P_1^*, \dots, P_{m^*}^* \vdash D^*$ be a subderivation of π, which has occurred before step i. Then to apply π^* in step i in π we must need a substitution $\lambda : \vec{x} \to \text{Lit}(\vec{x}) \cup \{0, 1\}$ is specified such that $P_j^*\restriction_\lambda$ is derived before step i in π, for all $j \leq m^*$. Then we can conclude $D^*\restriction_\lambda$ in step i.

Inherently we see two things here which are needed. The specification of the subderivation, which is to be applied, and the substitution which is to be used. For our purposes we will look at it in a very structured way to generalize better. We will not be using subderivations of a derivation, but for our purposes, a proof using lemmas would consist of a sequence of derivations performed in order, in which each derivation can use the conclusions of the previously derived derivations. Let $\vec{q}_1, \vec{q}_2, \dots, \vec{q}_k$ be disjoint variable sets, and let

$$\pi_1 : P_{1,1}, P_{1,2}, \dots, P_{1,m_1} \vdash D_1 \tag{3.1}$$

$$\pi_2 : P_{2,1}, P_{2,2}, \dots, P_{2,m_2} \vdash D_2 \tag{3.2}$$

$$\vdots$$

$$\pi_k : P_{k,1}, P_{k,2}, \dots, P_{k,m_k} \vdash D_k \tag{3.3}$$

be a sequence of derivations performed on variable sets $\vec{q}_1, \vec{q}_2, \dots, \vec{q}_k$ respectively. The final conclusion π_k is the one that should state $\bot \doteq 0 \geq 1$ if it is refutation of $\left\{P_{k,1}, \dots, P_{k,m_k}\right\}$. In these derivations we allow each step to be either a cutting planes rule as described in equations (2.5a) - (2.5f), or applying a previously derived lemma by the Lemma Application Rule, as described below:

Rule 3.1 (Lemma Application Rule)**.** Let $\pi_s = (E_1, E_2, \dots, E_{L_s})$ for a $2 \leq s \leq k$. In π_s we can derive the constraint E_j for a $j \leq L_s$, by application of a previous derivation π_t for a $t < s$, through specification of a substitution $\lambda : \vec{q}_t \to \text{Lit}(\vec{q}_s) \cup \{0, 1\}$ such that the following holds:

$$(i) \qquad D_t\restriction_\lambda \doteq E_j$$
$$(ii) \qquad \forall i \leq m_t \; \exists j_i < j \; : \; P_{t,i}\restriction_\lambda \doteq E_{j_i}$$

Let us break down the rule. To apply it using a previous derivation π_t we need the premises of π_t to have been derived already, however under a substitution. It also requires that a lemma π_t can only be applied if it derived before π_s where it is applied. That is to avoid the option that π_s could apply π_t and in π_t we could apply π_s. This kind of interdependency, would allow derivations to derive anything from anything, as π_t and π_s could just state the same thing and apply each other making it a "valid" derivation, but a circular argument.

We will note that in this formalization of proofs with lemmas, we need to know the all lemmas $\pi_1, \pi_2, \ldots, \pi_{k-1}$ before starting the main derivation π_k, which is not natural in an algorithmic context. But determining how an algorithm should in practice use derivations with lemmas is beyond the scope of this thesis.

From this rule we can see that for the first derivation π_1 we can only use the standard cutting planes rules. Moreover, we see that derivations can use each other recursively, as long as the derivation being applied is prior. This will let us define the *substitution depth* of a derivation to be 1 more than the maximum substitution depth of any of the lemmas that it applies. As π_1 cannot apply any lemmas we say that it has substitution depth 0. So in our setup above, π_k can at most have substitution depth $k-1$.
In our example above, say that π_j has length L_j for all $j \leq k$, then the total length of the proof would be $L = \sum_{j=1}^{k} L_j$. As hinted at earlier, instead of applying a lemma, using the lemma application rule, we could instead just have performed the derivation of the lemma. This means that we could just have inlined the previous derivation instead of applying it. If we were to inline the lemma derivations instead of applying them using rule 3.1, we would see that π_2 would at most have length $L_1 \cdot L_2$. This length can only be achieved if every single step in π_2 is a lemma application of π_1, which is quite unrealistic. By extending this argument we can easily upper bound the length of any derivation in which lemmas are inlined instead of applied. See that if π_3 was inlined in a similiar fashion as well, we would get that this derivation could not exceed a length of $L_3 \cdot (L_2 \cdot L_1)$. Generalizing this we see that for π_k inlined its length would be less than $L_k \cdot L_{k-1} \cdots L_1 \leq L^k$. This means that for a proof with lemmas with substitution depth $k-1$, the length of the proof can decrease with a $k'th$ root of the same proof with lemmas inlined. This also highlights the aim of allowing lemma applications, it will not introduce new ways of reasoning, but it can significantly shorten proofs.

Naively adding the rule 3.1 into VeriPB, is a bad idea for multiple reasons. First of all we have to ensure that the proof system is sound, and if we were to naively allow both rule 3.1 and the redundance-based strengthening rule, we would get an unsound proof system. This is shown in the example here:

Example 3.2. Say we are allowed to use the redundance-based strengthening rule inside lemma derivations, then we can derive the following two lemmas:

$$\pi_1 : (q_1 + q_2 \geq 1) \vdash (q_1 + q_2 \geq 2) \tag{3.4}$$
$$\pi_2 : (q_1 + q_2 \geq 1) \vdash (\bar{q}_1 + \bar{q}_2 \geq 1) \tag{3.5}$$

Both can be accomplished using the redundance-based strengthening rule, as we for π_1 can derive the final constraint in a single step using $\omega_1 = \{q_1 \to 1, q_2 \to 1\}$ which would mean the right hand side of (2.15) is always true, thus the derivation needed for redundance-based strengthening is trivial. The same can be said for π_2 using $\omega_2 = \{q_1 \to 1, q_2 \to 0\}$. But say we are allow to use these lemmas in our final derivation

$$\pi^* : (x + y \geq 1) \vdash \bot \tag{3.6}$$

As we could use π_1 first to achieve $(x + y \geq 2)$ by the substitution $\lambda = \{q_1 \to x_1, q_2 \to x_2\}$ and then use π_2 to achieve $(\overline{x} + \overline{y} \geq 1)$ by the same substitution. Then adding these two constraints would achieve $2 \geq 3$ which is equivalent to $\bot \doteq (0 \geq 1)$. Clearly this is an unsound derivation as the initial constraint can be satisfied, and thus combining these two rules naively can easily lead to some unsound derivations.

One could perhaps alter the definitions of either or both of these rules, such that soundness is preserved. But the second reason why adding the rule is undesirable is that we want to keep the proof system as simple as possible, as argued in the introduction. Instead, the approach of the thesis will be to achieve a similar kind of reasoning as applying lemmas, but without introducing a new rule of inference. This means that we wish to take a proof using lemmas, and translate it to a VeriPB proof. In the following chapters we will try to explain how this can be achieved, and what the caveats are.

The end goal, with respect to practical usage and implementation would then entail that a combinatorial solver would be allowed to produce a proof using lemmas with a rule such as 3.1, by specifying the lemma and the substitution. Then when the solver has finished, the proof log of the solver could then be converted to a VeriPB proof which could be verified. However, this would mean, that if VeriPB were to deem a proof incorrect, there is a chance that such a proof was indeed correct initially, but there is a bug in the conversion process. This is obviously not a desirable property, but since the other option is that the solvers which need this type of reasoning, would need all their lemmas inlined, and thus produce very long proofs, that might be impossible to verify in reasonable time, we still view it as the lesser of two evils.

4 Simulating a single lemma in VeriPB

The aim of this chapter is to explain and work through an example in which we will translate a derivation using a lemma twice, into a VeriPB proof. This example will give a good foundation for understanding the arguments and the method which we will use when translating derivations with lemmas into VeriPB proofs. The generalization of these arguments along with formal proofs for how we will simulate derivation using lemmas nested, will be given in chapter 5. However, due to the complicated nature of this translation in full generality, there will be some differences between these two chapters. The ideas given in this chapter will cover 90% of the ideas in the following chapter, but the most natural way to generalize this chapter will not work. We will explain why this is, and how it is fixed in the next chapter.

Let us introduce the example: the derivation with lemmas we wish to translate into a VeriPB proof is

$$\pi_1 : P_1, P_2, \ldots, P_{m_1} \vdash D_1 \tag{4.1}$$
$$\pi_2 : B_1, B_2, \ldots, B_{m_2} \vdash D_2 \tag{4.2}$$

where π_1 is performed on variable set $\vec{q}$ and π_2 is performed on variable set $\vec{x}$. Most importantly are the usages of rule 3.1, where π_1 is applied in π_2. In our example π_1 is applied twice in π_2, through the substitutions λ_1 and λ_2.

To simulate this in VeriPB, we are in essence just simulating π_2, this means that we create a VeriPB derivation $\pi^* : B_1, B_2, \ldots, B_{m_2} \vdash D_2$ thus we start with the premises in π_2. Any step which is not an application of π_1 will not be an issue, thus we will mainly focus on the application steps. In order to perform this translation, we will need the ability to "substitute" variables inside constraints, i.e. translating between constraint $C(\vec{q})$ and $C(\vec{q})\restriction_{\lambda_1}$. Notice that there are no rules in VeriPB which allow us to apply a substitution, thus we will need to have some other constraints which can help us achieve this. Let us say we had the following two constraints available

$$q_1 \leftrightarrow \lambda_1(q_1) \doteq \begin{cases} q_1 + \overline{\lambda_1(q_1)} \geq 1 & (4.3) \\ \overline{q_1} + \lambda_1(q_1) \geq 1 & (4.4) \end{cases}$$

where $q_1 \leftrightarrow \lambda_1(q_1)$ is also syntactic sugar that we will use. Then if we for example had the constraint

$$2q_1 + \overline{q_2} + q_3 \geq 2 \tag{4.5}$$

we could with the equation $\overline{q_1} + \lambda_1(q_1) \geq 1$ multiply it by 2 and add it to equation (4.5) to achieve:

$$2q_1 + \overline{q_2} + q_3 + 2\overline{q_1} + 2\lambda_1(q_1) \geq 2 + 2 \tag{4.6}$$
$$\doteq \quad 2\lambda_1(q_1) + \overline{q_2} + q_3 \geq 2 \tag{4.7}$$

in which we observe that q_1 has been substituted. This argument is generalized in the following proposition.

Proposition 4.1 (Implication substitution)**.** *Let $\vec{x}$ and $\vec{q}$ be disjoint variable sets. Let $\lambda : \vec{q} \to \mathrm{Lit}(\vec{x}) \cup \{0,1\}$ be a substitution, and let $C(\vec{q}) \doteq \sum a_i q_i^{\sigma_i} \geq A$. There exists cutting planes derivations*

$$\{C(\vec{q})\} \cup \{q_i \leftrightarrow \lambda(q_i) \mid \forall q_i \in \vec{q}\} \vdash C(\vec{q})\restriction_\lambda \tag{4.8}$$

and

$$\{C(\vec{q})\restriction_\lambda\} \cup \{q_i \leftrightarrow \lambda(q_i) \mid \forall q_i \in \vec{q}\} \vdash C(\vec{q}) \tag{4.9}$$

both with length at most $2|\vec{q}| + 1$.

Proof. The two derivations are actually exactly the same, and when one has been explained the other should be straightforward to see. Therefore, we will only show the derivation for equation (4.8).
Fix $q_{i^*} \in \vec{q}$, and for the literal $q_{i^*}^{\sigma_{i^*}}$ occurring in $C(\vec{q})$, note that we have

$$q_{i^*}^{1-\sigma_{i^*}} + \lambda(q_{i^*})^{\sigma_{i^*}} \geq 1 \tag{4.10}$$

as it corresponds to one of the constraints in $q_{i^*} \leftrightarrow \lambda(q_{i^*})$, regardless of the value of σ_{i^*}. Then by multiplying the constraint (4.10) with a_{i^*} and adding this to $C(\vec{q})$ we get

$$a_{i^*} q_{i^*}^{1-\sigma_{i^*}} + a_{i^*}\lambda(q_{i^*})^{\sigma_{i^*}} + a_{i^*} q_{i^*}^{\sigma_{i^*}} + \sum_{q_i \neq q_{i^*}} a_i q_i^{\sigma_i} \geq A + a_{i^*} \tag{4.11}$$

$$\doteq a_{i^*}\lambda(q_{i^*})^{\sigma_{i^*}} + \sum_{q_i \neq q_{i^*}} a_i q_i^{\sigma_i} \geq A \tag{4.12}$$

Since $a_{i^*} q_{i^*}^{\sigma_{i^*}}$ and $a_{i^*} q_{i^*}^{1-\sigma_{i^*}}$ cancel out, and leave a_{i^*} which is subtracted from both sides. Note that here $q_{i^*}^{\sigma_{i^*}}$ does not appear anymore and instead $\lambda(q_{i^*})^{\sigma_{i^*}}$ appears with the same coefficient.
If we were to perform this for each $q_i \in \vec{q}$, we would end up with the constraint

$$\left(\sum_{q_i \in \vec{q}} a_i \lambda(q_i)^{\sigma_i} \geq A \right) \doteq C(\vec{q})\restriction_\lambda \tag{4.13}$$

as desired. □

Now let us entertain the thought that in our example we could add $q_i \leftrightarrow \lambda_1(q_i)$ for all $q_i \in \vec{q}$, to our formula $F = \{B_j \mid \forall j \leq m_2\}$. Then in our VeriPB derivation, when we want to simulate the first lemma application, we know that we should already have derived $P_j\restriction_{\lambda_1}$ for all $j \leq m_1$, as that is required by the lemma application rule 3.1. Furthermore, we could without issue also introduce reification variables for P_j and then perform or "run" the derivation π_1 on $r\left(P_j\right) \implies P_j$ instead as explained in lemma 2.15, and we could derive $\bigwedge_{j=1}^{m_1} r\left(P_j\right) \implies \left(r\left(D_1\right) \geq 1\right)$. As the next proposition will show, we can use all of these constraints to achieve the desired constraint $D_1\restriction_{\lambda_1}$.

Proposition 4.2. *Let $\pi : P_1, \ldots, P_m \vdash D$ be a cutting planes derivation over variable set $\vec{q}$ which is disjoint from $\vec{x}$. Let $\lambda : \vec{q} \to \mathrm{Lit}(\vec{x}) \cup \{0, 1\}$ be a substitution. Then if we have a constraint database $\mathfrak{D}$ which contains*

$$\forall q_i \in \vec{q} \; : q_i \leftrightarrow \lambda(q_i) \tag{4.14}$$

$$\forall j \leq m \; : P_j\restriction_\lambda \tag{4.15}$$

$$\bigwedge_{j \leq m} r\left(P_j\right) \implies \left(r\left(D\right) \geq 1\right) \tag{4.16}$$

along with reification constraints for all the reification variables, then we can derive $D\restriction_\lambda$ and $r\left(D\restriction_\lambda\right) \geq 1$ using $O(|\vec{q}| \cdot m)$ cutting planes steps.

Proof. By using proposition 4.1 we can substitute the variables in $P_j\restriction_\lambda$ to derive P_j for all $j \leq m$. This is where the real cost lies, as it requires $O(|\vec{q}|)$ steps to substitute each variable in P_j and doing it for each $j \leq m$, means that this will in total require $O(|\vec{q}| \cdot m)$ steps. From P_j, as explained in proposition 2.12, we can thus derive $r\left(P_j\right) \geq 1$ for all $j \leq m$. For each of these we can remove them from the constraint in equation (4.16) as explained in proposition 2.13, thus achieving $r\left(D\right) \geq 1$. Again using proposition 2.12 we can thus derive D, which by again substituting variables using proposition 4.1 we get $D\restriction_\lambda$. Finally, we can derive the desired constraint $r\left(D\restriction_\lambda\right) \geq 1$ by proposition 2.12. □

This is the core idea behind how we will simulate a lemma application in VeriPB. Unfortunately, it is not as simple as this because to use substitution constraints $q_i \leftrightarrow \lambda(q_i)$, they first have to be added. Adding these constraints for the first substitution λ_1 can quite easily be done in our VeriPB derivation from the premises $B_1, \ldots, B_{m_2}$ using the redundance-based strengthening rule, as the variables q_i are fresh. However, once we want to introduce them for the second substitution λ_2 we are going to encounter a problem, because the q_i variables are not fresh anymore. Furthermore, if we were able to introduce these constraints for the second substitution we could introduce undesired dependencies between variables. Because from the constraints $q_i \leftrightarrow \lambda_1(q_i)$ and $q_i \leftrightarrow \lambda_2(q_i)$ we could derive the following

$$\lambda_1(q_i) \leftrightarrow \lambda_2(q_i) \doteq \begin{cases} \lambda_1(q_i) + \overline{\lambda_2(q_i)} \geq 1 \\ \overline{\lambda_1(q_i)} + \lambda_2(q_i) \geq 1 \end{cases} \tag{4.17}$$

which is not a constraint that should hold generally. For example if $\lambda_1(q_i) = x_1$ and $\lambda_2(q_i) = x_2$, then these constraints, would require that x_1 and x_2 have to take the same value under any satisfying assignment, which is purely an artifact of us introducing the substitution constraints and not a requirement of the original formula.

Instead of being able to substitute q_i and $\lambda_j(q_i)$ at any point, we instead we want to ensure that we only simulate the variable substitution when we are about simulate the lemma application. That is, we wish to have a setup essentially simulating: *"If π_1 has not been applied with λ_1 yet, then $q_i \leftrightarrow \lambda_1(q_i)$ should hold, and if the lemma has been applied with λ_1 but not yet with λ_2 then $q_i \leftrightarrow \lambda_2(q_i)$ should hold".*
This we achieve by reifying the conclusions we wish to achieve, namely $D_1\restriction_{\lambda_1}$ and $D_1\restriction_{\lambda_2}$, and then having the following constraints

$$\overline{r\left(D_1\restriction_{\lambda_1}\right)} \implies q_i \leftrightarrow \lambda_1(q_i) \doteq \begin{cases} \overline{r\left(D_1\restriction_{\lambda_1}\right)} \implies \left(q_i + \overline{\lambda_1(q_i)} \geq 1\right) & (4.18) \\ \overline{r\left(D_1\restriction_{\lambda_1}\right)} \implies (\overline{q_i} + \lambda_1(q_i) \geq 1) & (4.19) \end{cases}$$

$$r\left(D\restriction_{\lambda_1}\right) \wedge \overline{r\left(D\restriction_{\lambda_2}\right)} \implies q_i \leftrightarrow \lambda_2(q_i) \doteq \begin{cases} r\left(D\restriction_{\lambda_1}\right) \wedge \overline{r\left(D\restriction_{\lambda_2}\right)} \implies \left(q_i + \overline{\lambda_1(q_i)} \geq 1\right) & (4.20) \\ r\left(D\restriction_{\lambda_1}\right) \wedge \overline{r\left(D\restriction_{\lambda_2}\right)} \implies (\overline{q_i} + \lambda_1(q_i) \geq 1) & (4.21) \end{cases}$$

Combining our syntax in the most convenient way. When performing the derivation in proposition 4.1 with equation (4.18) and (4.19) instead of $q_i \leftrightarrow \lambda_1(q_i)$, we will derive

$$\overline{r\left(D_1\restriction_{\lambda_1}\right)} \implies \left(r\left(D_1\restriction_{\lambda_1}\right) \geq 1\right) \tag{4.22}$$

as we in proposition 2.16 explained how "conditions" can be added to some premises in a derivation and then be carried all the way through the derivation. If we inspect equation (4.22) we see that it is exactly $2r\left(D_1\restriction_{\lambda_1}\right) \geq 1$, which can be saturated to get $r\left(D_1\restriction_{\lambda_1}\right) \geq 1$. Thus, we are still able to use these constraints, to substitute variables and "apply" the lemma. Now we know the variable $r\left(D_1\restriction_{\lambda_1}\right)$ must be true, and by proposition 2.13 we can remove it from the implications in (4.20) and (4.21) to get

$$\overline{r\left(D_1\restriction_{\lambda_2}\right)} \implies q_i \leftrightarrow \lambda_2(q_i) \tag{4.23}$$

Then we quite recursively have the same setup when we need to simulate the next lemma application. Thus, the complete translation of the derivation π_1 and π_2 into a VeriPB proof, would look like this:

- First add reifications for $D\restriction_{\lambda_1}$ and $D\restriction_{\lambda_2}$, by the redundance-based strengthening rule, as explained in proposition 2.11.
- Then add (4.18) and (4.19) for each q_i by the redundance-based strengthening rule. We are not going to cover this in details here, but mainly sketch the idea on how this can be done. First equation (4.18) is added by witnessing on $\omega = \{q_i \to \lambda_1(q_i)\}$, which satisfies the constraint itself, and as q_i is fresh no other constraints are affected. Secondly the constraint (4.19) is added by the same witness $\omega = \{q_i \to \lambda_1(q_i)\}$, in which both (4.18) and (4.19) are satisfied, and no other constraints are affected. Then we add equation (4.20) by the witness $\omega = \{q_i \to \lambda_2(q_i)\}$, where the constraint itself is satisfied, and only the constraints $q_i \leftrightarrow \lambda_1(q_i)$ for the same q_i, are affected. To derive these, we use the fact that in the redundance-based strengthening rule to add a constraint C we have to show

  $$F \cup \{\neg C\} \vdash (F \cup \{C\})\restriction_{\omega} \tag{4.24}$$

  which means that in our example we have

  $$\neg\left(r\left(D\restriction_{\lambda_1}\right) \wedge \overline{r\left(D\restriction_{\lambda_2}\right)} \implies \left(q_i + \overline{\lambda_2(q_i)} \geq 1\right)\right) \tag{4.25}$$

  $$\doteq \neg\left(\overline{r\left(D\restriction_{\lambda_1}\right)} + r\left(D\restriction_{\lambda_2}\right) + q_i + \overline{\lambda_2(q_i)} \geq 1\right) \tag{4.26}$$

  $$\doteq r\left(D\restriction_{\lambda_1}\right) + \overline{r\left(D\restriction_{\lambda_2}\right)} + \overline{q_i} + \lambda_2(q_i) \geq 4 \tag{4.27}$$

exactly as showcased in example 2.5. Specifically because (4.20) is in disjunctive form, and we have the negation of it, we can by proposition 2.6 derive $r\left(D\restriction_{\lambda_1}\right) \geq 1$, from which both constraints in $\overline{r\left(D\restriction_{\lambda_1}\right)} \implies q_i \leftrightarrow \lambda_1(q_i)$ are literal axiom implied. This means that (4.20) can be added by the redundance-based strengthening rule, and the argument for why (4.21) can be added is exactly the same.

- Add reifications for P_j for all $j \leq m_1$ by the redundance-based strengthening rule, again as given in proposition 2.11. Then using the constraints $r\left(P_j\right) \implies P_j$ we can "run" π_1 as explained in lemma 2.15 to derive $\bigwedge_{j \leq m_1} r\left(P_j\right) \implies r\left(D_1\right)$.
- Now perform π_2 as usual until the first lemma application happens, and then use proposition 5.2 along with proposition 2.16 to derive $r\left(D_1\restriction_{\lambda_1}\right) \geq 1$, and then also $D_1\restriction_{\lambda_1}$ by proposition 2.12.
- Use $r\left(D_1\restriction_{\lambda_1}\right) \geq 1$ to "clean up" in the substitution constraints in (4.20) and (4.21) to get (4.23).
- Keep performing π_2 until the second lemma application happens, and then again use proposition 5.2 along with proposition 2.16 to derive $r\left(D_1\restriction_{\lambda_2}\right) \geq 1$, and then also $D_1\restriction_{\lambda_2}$.
- Now the rest of π_2 can be performed as usual, and we end up concluding D_2.

This exact method is not going to work in full generality as we will show in the following chapter, but the core idea and derivations will stay the same.

5 Simulating nested lemmas

In this chapter we will generalize the method explained in the previous chapter to handle derivations with lemmas such as

$$\pi_1 : P_{1,1}, P_{1,2}, \ldots, P_{1,m_1} \vdash D_1 \tag{5.1}$$
$$\pi_2 : P_{2,1}, P_{2,2}, \ldots, P_{2,m_2} \vdash D_2 \tag{5.2}$$
$$\vdots \tag{5.3}$$
$$\pi_k : P_{k,1}, P_{k,2}, \ldots, P_{k,m_k} \vdash D_k \tag{5.4}$$

where π_t is derived on variable set $\vec{q}_t$, and $\pi_1, \pi_2, \ldots, \pi_k$ are disjoint. At first we will prove that we can simulate π_k based on a few assumptions. One assumption is that we have already "performed" $\pi_1, \pi_2, \ldots, \pi_{k-1}$ on reified premises, that is we have already derived

$$\bigwedge_{j=1}^{m_s} r\left(P_{s,j}\right) \implies \left(r\left(D_s\right) \geq 1\right) \tag{5.5}$$

for all $1 \leq s \leq k-1$. To help ourselves, as this some notation that we will use often, we will for any constraint C instead write

$$\mathrm{Imp}_s(C) \doteq \left(\bigwedge_{j=1}^{m_s} r\left(P_{s,j}\right) \implies C\right) \tag{5.6}$$

to capture that the constraint C is implied by the premises of π_s. The secondary assumption is that when we "ran" $\pi_1, \pi_2, \ldots, \pi_{k-1}$ we also made sure that they cleaned up in the constraints needed for variable substitution. This means that when we want to simulate all applications of π_1 in π_k we have

$$\overline{r\left(D_1\restriction_{\lambda_{1,1}}\right)} \implies q_i \leftrightarrow \lambda_{1,1}(q_i) \tag{5.7}$$

$$r\left(D_1\restriction_{\lambda_{1,1}}\right) \wedge \overline{r\left(D_1\restriction_{\lambda_{1,2}}\right)} \implies q_i \leftrightarrow \lambda_{1,2}(q_i) \tag{5.8}$$

$$\vdots$$

where $\lambda_{1,j}$ is the substitution for the j'th application of π_1 in π_k. Let $J(\pi_s)$ be the total number of applications of π_s in π_k. Then we assume that we for all $s < k$ and all $j \leq J(\pi_s)$ have

$$\bigwedge_{j'<j} r\left(D_s\restriction_{\lambda_{s,j'}}\right) \wedge \overline{r\left(D_s\restriction_{\lambda_{s,j}}\right)} \implies q_i \leftrightarrow \lambda_{s,j}(q_i) \tag{5.9}$$

where $\lambda_{s,j}$ is the substitution for the j'th application of π_s in π_k. Using the two assumptions that we have (5.5) and (5.9), we will prove that we can simulate π_k in cutting planes.

Proposition 5.1. *Let $k \geq 2$ and let*

$$\begin{aligned}
\pi_1 &: P_{1,1}, P_{1,2}, \ldots, P_{1,m_1} \vdash D_1 \\
\pi_2 &: P_{2,1}, P_{2,2}, \ldots, P_{2,m_2} \vdash D_2 \\
&\vdots \\
\pi_k &: P_{k,1}, P_{k,2}, \ldots, P_{k,m_k} \vdash D_k
\end{aligned}$$

be a derivation using lemmas performed on variable sets $\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_k$ respectively, which are disjoint. Let $\lambda_{s,j}$ be the variable substitution for the j'th application of π_s in π_k for $s < k$, for a total of $J(\pi_s)$ applications. If we have a constraint database $\mathfrak{D}$ which contains the premises of π_k as well as

$$\forall s < k \ : \ \mathrm{Imp}_s(D_s) \doteq \left(\bigwedge_{j=1}^{m_s} r\left(P_{s,j}\right) \implies D_s\right) \tag{5.10}$$

$$\forall s < k \ \forall j \leq J(\pi_s) \ : r\left(D_s\restriction_{\lambda_{s,j}}\right) \iff D_s\restriction_{\lambda_{s,j}} \tag{5.11}$$

$$\forall s < k \ \forall j \leq J(\pi_s) \ \forall q_i \in \vec{q}_s : \bigwedge_{j'<j} r\left(D_s\restriction_{\lambda_{s,j'}}\right) \wedge \overline{r\left(D_s\restriction_{\lambda_{s,j}}\right)} \implies q_i \leftrightarrow \lambda_{s,j}(q_i) \tag{5.12}$$

along with reification variables for all premises across $\pi_1, \pi_2, \ldots, \pi_k$. Then we can derive D_k in cutting planes.

Proof. All steps in π_k which are standard cutting planes rule can be performed as usual, since we have the premises $P_{k,1}, \ldots, P_{k,m_k}$. What we need to show is that we can translate each lemma application in π_k to a cutting planes derivation. I.e. that we can derive $D_s\restriction_{\lambda_{s,j}}$ for all substitutions $\lambda_{s,j}$. We will do so by induction on the number of lemma applications $J(\pi_1) + J(\pi_2) + \cdots + J(\pi_{k-1})$ in π_k.

The base case
For the first lemma application there exists an $s < k$, such that $\lambda_{s,1}$ is the substitution for the lemma application. By assumption of the lemma application rule, we must then have derived

$$P_{s,1}\restriction_{\lambda_{s,1}}, P_{s,2}\restriction_{\lambda_{s,1}}, \dots, P_{s,m_s}\restriction_{\lambda_{s,1}} \tag{5.13}$$

previously in π_k. By using these as well as $\text{Imp}_s(D_s)$ and

$$\forall q_i \in \vec{q}_k \ : \overline{r\left(D_s\restriction_{\lambda_{s,1}}\right)} \implies q_i \leftrightarrow \lambda_{s,1}(q_i) \tag{5.14}$$

we can by proposition 5.2 and 2.16 derive $\overline{r\left(D_s\restriction_{\lambda_{s,1}}\right)} \implies \left(r\left(D_s\restriction_{\lambda_{s,1}}\right) \geq 1\right)$ which when saturated gives $r\left(D_s\restriction_{\lambda_{s,1}}\right) \geq 1$, and can then be used to derive $D_s\restriction_{\lambda_{s,1}}$.

The induction step
Say we have reached the application using the substitution $\lambda_{s,j}$ for a $s < k$ and $j \leq J(\pi_s)$. By our induction hypothesis we have already derived $D_s\restriction_{\lambda_{s,j'}}$ for all $j' < j$ regardless of the value of s. In fact, we already have derived $r\left(D_s\restriction_{\lambda_{s,j'}}\right) \geq 1$ as well, but this could also quickly be derived from $D_s\restriction_{\lambda_{s,j'}}$. By repeated use of proposition 2.13 we can from

$$\bigwedge_{j'<j} r\left(D_s\restriction_{\lambda_{s,j'}}\right) \wedge \overline{r\left(D_s\restriction_{\lambda_{s,j}}\right)} \implies q_i \leftrightarrow \lambda_{s,j}(q_i) \tag{5.15}$$

remove $r\left(D_s\restriction_{\lambda_{s,j'}}\right)$ for all $j' < j$ since we know that $r\left(D_s\restriction_{\lambda_{s,j'}}\right) \geq 1$ and derive

$$\overline{r\left(D_s\restriction_{\lambda_{s,j}}\right)} \implies q_i \leftrightarrow \lambda_{s,j}(q_i) \tag{5.16}$$

This corresponds to the same situation as in the base case, as we by proposition 5.2 and 2.16 can use $\text{Imp}_s(D_s)$ and (5.16) along with $P_{s,1}\restriction_{\lambda_{s,j}}, P_{s,2}\restriction_{\lambda_{s,j}}, \dots, P_{s,m_s}\restriction_{\lambda_{s,j}}$ which have already been derived by assumption of the lemma application rule, to derive $D_s\restriction_{\lambda_{s,j}}$ and $r\left(D_s\restriction_{\lambda_{s,j}}\right) \geq 1$.

□

This derivation is thus a completely standard cutting planes derivation, and by proposition 2.16, we could perform this on reified premises $r\left(P_{k,j}\right) \implies P_{k,j}$ for all $j \leq m_k$, to at the en derive $\text{Imp}_k(D_k)$. Now the start of an inductive proof should start to emerge, as we see that we can "run" π_k on reified premises, and partly achieve the assumption we would need if we were to have a π_{k+1}. The other part missing, is the assumption that π_k also "cleans up" in the constraints for variable substitution in π_{k+1} and further derivations, if there are any. The following example will highlight what exactly we mean: Say we have π_1, π_2, π_3 which is a derivation with lemmas with conclusions D_1, D_2, D_3 respectively. If π_1 is applied once in π_2 with $\lambda_{1,1}$ and once in π_3 with $\lambda_{1,2}$, then we will need our substitution constraints for these applications to be

$$\overline{r\left(D_1\restriction_{\lambda_{1,1}}\right)} \implies q_i \leftrightarrow \lambda_{1,1}(q_i) \tag{5.17}$$

$$r\left(D_1\restriction_{\lambda_{1,1}}\right) \wedge \overline{r\left(D_1\restriction_{\lambda_{1,2}}\right)} \implies q_i \leftrightarrow \lambda_{1,2}(q_i) \tag{5.18}$$

When we say "clean up", what we mean is that when running π_2 on reified premises, we want it to remove all the variable $r\left(D_1\restriction_{\lambda_{1,1}}\right)$ from all substitution constraints needed in subsequent

derivations, such that when we want to run π_3 (possibly on reified premises) there are no references to previous lemma applications. The problem is that when we run π_2 on reified premises we never actually derive $D_1\restriction_{\lambda_{1,1}}$, instead we would derive $\mathrm{Imp}_2(D_1\restriction_{\lambda_{1,1}})$, as explained in lemma 2.15, from which we **cannot** conclude that $r\left(D_1\restriction_{\lambda_{1,1}}\right) \geq 1$.

This is the last hurdle that we need to solve before we can prove the translation in full generality. The solution is in fact, quite intuitive. If we are not deriving $D_1\restriction_{\lambda_{1,1}}$ in π_2, but instead $\mathrm{Imp}_2(D_1\restriction_{\lambda_{1,1}})$, then replace the condition in the substitution constraints from (5.17) and (5.18) with

$$\overline{r\left(\mathrm{Imp}_1(D_1\restriction_{\lambda_{1,1}})\right)} \implies q_i \leftrightarrow \lambda_{1,1}(q_i) \tag{5.19}$$

$$r\left(\mathrm{Imp}_1(D_1\restriction_{\lambda_{1,1}})\right) \wedge \overline{\mathrm{Imp}_2(r\left(D_1\restriction_{\lambda_{1,2}}\right))} \implies q_i \leftrightarrow \lambda_{1,2}(q_i) \tag{5.20}$$

The fact that these can still be used to perform the lemma applications is just an argument combining proposition 5.2 and proposition 2.16, and the formal statement along with proof is given in the following proposition.

Proposition 5.2. *$\pi : P_1, \ldots, P_m \vdash D$ be a cutting planes derivation over variable set $\vec{q}$ which is disjoint from $\vec{x}$. Let $\lambda : \vec{q} \to \mathrm{Lit}(\vec{x}) \cup \{0,1\}$ be a substitution, and let*

$$\mathrm{Imp}(C) \doteq \left(\bigwedge_{j=1}^{m_2} r\left(B_j\right) \implies C\right)$$

denote that C is conditionally implied by some premises $B_1, \ldots, B_{m_2}$. If we have a constraint database $\mathfrak{D}$ which contains

$$\forall q_i \in \vec{q} : \overline{r\left(\mathrm{Imp}(D\restriction_\lambda)\right)} \implies q_i \leftrightarrow \lambda(q_i) \tag{5.21}$$

$$\forall j \leq m : \mathrm{Imp}(P_j\restriction_\lambda) \tag{5.22}$$

$$\bigwedge_{j \leq m} r\left(P_j\right) \implies \left(r\left(D\right) \geq 1\right) \tag{5.23}$$

along with reification constraints for all the reification variables, then we can derive $r\left(\mathrm{Imp}(r\left(D\restriction_\lambda\right))\right) \geq 1$ using $O(|\vec{q}| \cdot m)$ cutting planes steps.

Proof. We notice that the only difference between this setup and the one in proposition 4.1, is that the constraints in (5.21) and (5.22) are implied conditionally. Thus, as the derivation given in proposition 4.1 is a cutting planes derivation, we can by proposition 2.16 perform the same derivation on these conditionally implied constraints instead, and thus derive

$$\bigwedge_{j \leq m_2} r\left(B_j\right) \wedge \overline{r\left(\mathrm{Imp}(D\restriction_\lambda)\right)} \implies \left(r\left(D\restriction_\lambda\right) \geq 1\right) \tag{5.24}$$

this we can by proposition 2.12 use to get

$$\bigwedge_{j \leq m_2} r\left(B_j\right) \wedge \overline{r\left(\mathrm{Imp}(D\restriction_\lambda)\right)} \implies D\restriction_\lambda \tag{5.25}$$

but in our syntax this corresponds exactly to

$$\overline{r\left(\mathrm{Imp}(D\restriction_\lambda)\right)} \implies \left(\bigwedge_{j\leq m_2} r\left(B_j\right) \wedge \implies D\restriction_\lambda\right) \tag{5.26}$$

$$\doteq \overline{r\left(\mathrm{Imp}(D\restriction_\lambda)\right)} \implies \mathrm{Imp}(D\restriction_\lambda) \tag{5.27}$$

here we again can use proposition 2.12 to get

$$\overline{r\left(\mathrm{Imp}(D\restriction_\lambda)\right)} \implies \left(r\left(\mathrm{Imp}(D\restriction_\lambda)\right) \geq 1\right) \tag{5.28}$$

$$\doteq r\left(\mathrm{Imp}(D\restriction_\lambda)\right) + r\left(\mathrm{Imp}(D\restriction_\lambda)\right) \geq 1 \tag{5.29}$$

which when saturated gives us the desired conclusion $r\left(\mathrm{Imp}(D\restriction_\lambda)\right) \geq 1$

□

Now we only have one thing remaining to prove. The fact that we can actually add the substitution constraints at the very start of the translation. Then finally we will in theorem 1 show that we can combine all these propositions, and thereby translate any derivation with lemmas to a VeriPB derivation.

5.1 Substition of variables

The first proposition will allow us to add the necessary substitution constraints for all applications of a single lemma, which is across all other derivations.

Proposition 5.3. *Let*

$$\begin{aligned}
\pi_1 &: P_{1,1}, \ldots, P_{1,m_1} \vdash D_1 \\
&\vdots \\
\pi_k &: P_{k,1}, \ldots, P_{k,m_k} \vdash D_k
\end{aligned}$$

be a derivation with lemmas on respectively the variable sets $\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_k$ which are all disjoint. Let $J(\pi_s) \in \mathbb{N}_0$ denote the total number of times π_s is being applied across $\pi_{s+1}, \ldots, \pi_k$. Furthermore, let $\lambda_j : \vec{q}_s \to \mathrm{Lit}(\vec{q}_{t_j}) \cup \{0, 1\}$ be the variable substitution for the j'th lemma application of π_s which is done in π_{t_j}, for $1 \leq s < t_j \leq k$. Recall the notation

$$\mathrm{Imp}_t(C) \doteq \left(\bigwedge_{j=1}^{m_t} r\left(P_{t,j}\right) \implies C\right) \tag{5.30}$$

for the constraint capturing that the premises of π_t implies C. If the constraint database $\mathfrak{D}$ of currently derived constraints does not contain any constraint with variables from $\vec{q}_s$, and if $\mathfrak{D}$ contains the reification variables for all premises across $\pi_{s+1}, \pi_{s+2}, \ldots, \pi_k$. Then for all applications of π_s, i.e. for all $j \leq J(\pi_s)$ we can add

$$r\left(\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j})\right) \iff \mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j}) \tag{5.31}$$

as well as

$$\bigwedge_{j^*<j} r\left(\mathrm{Imp}_{t_{j^*}}(D_s\restriction_{\lambda_{j^*}})\right) \wedge \overline{r\left(\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j})\right)} \implies (q_i \leftrightarrow \lambda_j(q_i)) \tag{5.32}$$

for all $q_i \in \vec{q}_s$.

Proof. We can add the constraints in (5.31) for all $j \leq J(\pi_s)$ by proposition 2.11, notice also that these constraints are over variables in $\vec{q}_{t_j}$ for $t_j > s$ and reification variables of the premises in π_{t_j}. This means that $\vec{q}_s$ is still fresh
Now recall that equation (5.32) is syntactic sugar [2] for the following two constraints:

$$C_{j,i} \doteq \left(\sum_{j^*<j} \overline{r\left(\mathrm{Imp}_{t_{j^*}}(D_s\restriction_{\lambda_{j^*}})\right)} + r\left(\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j})\right) + q_i + \overline{\lambda_j(q_i)} \geq 1\right) \tag{5.33}$$

$$B_{j,i} \doteq \left(\sum_{j^*<j} \overline{r\left(\mathrm{Imp}_{t_{j^*}}(D_s\restriction_{\lambda_{j^*}})\right)} + r\left(\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j})\right) + \overline{q_i} + \lambda_j(q_i) \geq 1\right) \tag{5.34}$$

So we need to prove that the constraints $C_{j,i}$ and $B_{j,i}$ can be added to $\mathfrak{D}$ for all $j \leq J(\pi_s)$, and all $q_i \in \vec{q}_s$. We will prove that we can add them for each $q_i \in \vec{q}_s$ based on induction on j.

Base case $j = 1$*:*
Fixing $q_i \in \vec{q}_s$, we will first add the constraint

$$C_{1,i} \doteq \left(r\left(\mathrm{Imp}_{t_1}(D_s\restriction_{\lambda_1})\right) + q_i + \overline{\lambda_1(q_i)} \geq 1\right) \tag{5.35}$$

to $\mathfrak{D}$. We do so by the redundance-based strengthening rule, using the witness $\omega = \{q_i \to \lambda_j(q_i)\}$. Recalling the rule we need to show that

$$\mathfrak{D} \cup \{\neg C_{1,i}\} \vDash (\mathfrak{D} \cup \{C_{1,i}\})\restriction_\omega \tag{5.36}$$

Notice that $C_{1,i}\restriction_\omega$ is satisfied, thus as we have shown in proposition 2.7, it is trivial to derive. As we assumed that $\mathfrak{D}$ does not contain any constraints with q_i, none of the constraints in $\mathfrak{D}$ are affected by ω, thus the implication $\mathfrak{D} \vDash \mathfrak{D}\restriction_\omega$ is trivial, as they are equal.

This means that the derivation neccesary for satisfying the condition in equation (5.36) is quite trivial, and we can add $C_{1,i}$ to $\mathfrak{D}$. Doing this for each $q_i \in \vec{q}_t$ can be done, as they all witness on differenct q_i, which means each witness will not affect any other constraint.
Now we have the constraint database

$$\mathfrak{D}^* := \mathfrak{D} \cup \{C_{1,i} \mid \forall q_i \in \vec{q}_s\} \tag{5.37}$$

to which we will again fix $q_i \in \vec{q}_s$ and add the constraint

$$B_{1,i} \doteq \left(r\left(\mathrm{Imp}_{t_1}(D_s\restriction_{\lambda_1})\right) + \overline{q_i} + \lambda_1(q_i) \geq 1\right) \tag{5.38}$$

[2] At this point it perhaps doesn't feel like sugar anymore, but resembles more the feeling of chewing a lemon, however it is still easier to digest this syntax than if we were to write out everything

again by the same witness $\omega = \{q_i \to \lambda_1(q_i)\}$, which means that $\omega(\overline{q_i}) = \overline{\omega(q_i)} = \overline{\lambda_1(q_i)}$. Again we notice that both $C_{1,i}\restriction_\omega$ and $B_{1,i}\restriction_\omega$ are satisfied under this witness. This means that both of these are trivial to derive, and as argued before none of the constraints in $\mathfrak{D}$ are affected by ω, as well as the other constraints $C_{1,i'}$ and $B_{1,i'}$ for different $q_{i'} \neq q_i$ in $\vec{q}_s$. Finally we conclude that both $C_{1,i}$ and $B_{1,i}$ can be added to $\mathfrak{D}$ for each $q_i \in \vec{q}_t$, therefore we have as desired that we can achieve $\mathfrak{D}_1 := \mathfrak{D} \cup \{C_{1,i} \mid \forall q_i \in \vec{q}_s\} \cup \{B_{1,i} \mid \forall q_i \in \vec{q}_s\}$. Now concluding that we can add equation (5.32) for $j = 1$.

Inductive step $j \geq 2$:
Using the definition as above we recursively define:

$$\mathfrak{D}_j := \mathfrak{D}_{j-1} \cup \{C_{j,i} \mid \forall q_i \in \vec{q}_s\} \cup \{B_{j,i} \mid \forall q_i \in \vec{q}_s\} \tag{5.39}$$

for $j \leq J(\pi_s)$. By the induction hypothesis we currently have the constraint database $\mathfrak{D}_{j-1}$, and wish to achieve $\mathfrak{D}_j$.

Again let $q_i \in \vec{q}_s$ be fixed. To introduce the constraint $C_{j.i}$, we will by the very same manner as previously use the redundance-based strengthening rule with the witness $\omega = \{q_i \to \lambda_j(q_i)\}$. Again we notice that the constraint itself is satisfied under this witness, and can be derived trivially. The only other constraints in $\mathfrak{D}_{j-1}$ which contain q_i are $C_{j^*,i}$ and $B_{j^*,i}$ for all $j^* < j$, thus the necessary condition to add C_j, i to $\mathfrak{D}_{j-1}$ is to show that

$$\mathfrak{D}_{j-1} \cup \{\neg C_{j,i}\} \vDash \{C_{j^*,i}\restriction_\omega \mid \forall j^* < j\} \cup \{B_{j^*,i}\restriction_\omega \mid \forall j^* < j\} \tag{5.40}$$

Lets fix $j^* < j$ and derive $C_{j^*,i}\restriction_\omega$ and $B_{j^*,i}\restriction_\omega$ which we will do by using $\neg C_{j,i}$. Notice that $C_{j,i}$ is a constraint in disjunctive form, thus we have that

$$\neg C_{j,i} \doteq \left(\sum_{j'<j} \overline{r\left(\mathrm{Imp}_{t_{j'}}(D_s\restriction_{\lambda_{j'}})\right)} + r\left(\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j})\right) + \overline{q_i} + \lambda_j(q_i) \geq j + 2 \right) \tag{5.41}$$

and from this we can derive $\overline{r\left(\mathrm{Imp}_{t_{j'}}(D_s\restriction_{\lambda_{j'}})\right)} \geq 1$ for any $j' < j$, as explained in proposition 2.6. Specifically we can derive $\overline{r\left(\mathrm{Imp}_{t_{j^*}}(D_s\restriction_{\lambda_{j^*}})\right)} \geq 1$ from which we can simply add literal axioms until we get

$$C_{j^*,i}\restriction_\omega \doteq \left(\sum_{j'<j^*} \overline{r\left(\mathrm{Imp}_{t_{j'}}(D_s\restriction_{\lambda_{j'}})\right)} + r\left(\mathrm{Imp}_{t_{j^*}}(D_s\restriction_{\lambda_{j^*}})\right) + \lambda_j(q_i) + \overline{\lambda_{j^*}(q_i)} \geq 1 \right) \tag{5.42}$$

Likewise we could add literal axioms to $\overline{r\left(\mathrm{Imp}_{t_{j^*}}(D_s\restriction_{\lambda_{j^*}})\right)} \geq 1$ until we get

$$B_{j^*,i}\restriction_\omega \doteq \left(\sum_{j'<j^*} \overline{r\left(\mathrm{Imp}_{t_{j'}}(D_s\restriction_{\lambda_{j'}})\right)} + r\left(\mathrm{Imp}_{t_{j^*}}(D_s\restriction_{\lambda_{j^*}})\right) + \overline{\lambda_j(q_i)} + \lambda_{j^*}(q_i) \geq 1 \right) \tag{5.43}$$

As j^* was arbitrary, this derivation can be done for all $j^* < j$. Thus we have

$$\{\neg C_{j,i}\} \vdash \{C_{j^*,i}\restriction_\omega \mid \forall j^* < j\} \cup \{B_{j^*,i}\restriction_\omega \mid \forall j^* < j\}$$

which means the criteria for the redundance-based strengthening rule is satisfied, and $C_{j,i}$ can be added to $\mathfrak{D}_{j-1}$.

We add $B_{j,i}$ by exactly the same argument. We let $\omega = \{q_i \to \lambda_j(q_i)\}$ where both $C_{j,i}\restriction_\omega$ and $B_{j,i}\restriction_\omega$ are satisfied and by exactly the same argument as above we have that

$$\{\neg B_{j,i}\} \vdash \{C_{j^*,i}\restriction_\omega \mid \forall j^* < j\} \cup \{B_{j^*,i}\restriction_\omega \mid \forall j^* < j\}$$

by literal axiom implication. Again we can do this for each $q_i \in \vec{q}_t$ as the witness will not interfere with $C_{j,i'}$ and $B_{j,i'}$ for $q_{i'} \neq q_i$. Finally we conclude that we can get the formula $\mathfrak{D}_j$. □

The main assumption that made proposition 5.3 possible was that there were no constraints in $\mathfrak{D}$ which contained variables from $\vec{q}_s$. To clarify how this is going to be achieved, first we will clarify which variables are needed to add the reification variables we need for proposition 5.3. Let us unpack the reification variable $r\left(\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j})\right)$ To introduce it we need

$$r\left(\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j})\right) \implies \mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j}) \tag{5.44}$$

$$r\left(\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j})\right) \impliedby \mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j}) \tag{5.45}$$

So for these to be added we need the variables occuring in $\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j})$. Now we unpack the next layer of syntax and see

$$\mathrm{Imp}_{t_j}(D_s\restriction_{\lambda_j}) \doteq \left(\bigwedge_{l \leq m_{t_j}} r\left(P_{t_j,l}\right) \implies D_s\restriction_{\lambda_j}\right) \tag{5.46}$$

This means we need variables in $D_s\restriction_{\lambda_j}$ which are $\vec{q}_{t_j}$, and recall that $t_j > s$. We also need the reification variables for the premises $r\left(P_{t_j,l}\right)$ of π_{t_j}. This means that when we want to add the reification constraints in (5.44) and (5.45) we have to have already setup the substitution constraints for π_{t_j}. If we were to do it for π_s first, which is applied in π_{t_j}, the variables in $\vec{q}_{t_j}$ would not be fresh anymore, and we could not apply proposition 5.3. If we do it in "reverse" order, i.e. setup all substitution constraints for π_k first (which are none, as π_k can never being applied anywhere), then the variables in π_{k-1} are still fresh, and we can still apply proposition 5.3 for π_{k-1}. This is the main idea, that lets us add the substitution constraints for all applications across all derivations.

Lemma 5.4. *Let*

$$\begin{aligned}
&\pi_1 : P_{1,1}, \ldots, P_{1,m_1} \vdash D_1\\
&\quad\vdots\\
&\pi_k : P_{k,1}, \ldots, P_{k,m_k} \vdash D_k
\end{aligned}$$

be a sequence of derivations with lemmas on respectively the variable sets $\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_k$ which are all disjoint. Let $\lambda_{s,j} : \vec{q}_s \to \vec{q}_{t_{s,j}}$ be the variable substitution for the j'th lemma application of π_s which is done in $\pi_{t_{s,j}}$, for $1 \leq s < t_{s,j} \leq k$, for a total of $J(\pi_s) \in \mathbb{N}_0$ applications of π_s.

Let $F \doteq \left\{P_{k,1}, P_{k,2}, \ldots, P_{k,m_k}\right\}$. *To* F *we can then add reification variables for all premises and conclusions in* $\pi_1, \pi_2, \ldots, \pi_k$, *namely the constraints*

$$\forall s \leq k;\ \ \forall l \leq m_s\ :\ r\left(P_{s,l}\right) \iff P_{s,l}$$
$$\forall s \leq k\ :\ r\left(D_s\right) \iff D_s$$

Furthermore we can also add reifications for the conclusions of all lemma applications, which are the constraints

$$\forall s \leq k;\ \ \forall j \leq J(\pi_s)\ :\ r\left(D_s\restriction_{\lambda_{s,j}}\right) \iff D_s\restriction_{\lambda_{s,j}} \tag{5.47}$$

and the substitution constraints for all applications, which are

$$\bigwedge_{j^*<j} r\left(\mathrm{Imp}_{t_{s,j^*}}(D_s\restriction_{\lambda_{s,j^*}})\right) \wedge \overline{r\left(\mathrm{Imp}_{t_{s,j}}(D_s\restriction_{\lambda_{s,j}})\right)}\,(q_{s,i} \leftrightarrow \lambda_{s,j}(q_{s,i})) \tag{5.48}$$

for all $s \leq k$, $j \leq J(\pi_s)$ *and* $q_{s,i} \in \vec{q}_s$.

Proof. To show all of these can be added we will do so by induction on $s = k, k-1, k-2, \ldots, 1$. In each step we let $\mathfrak{D}_s$ be the current constraint database and the induction hypothesis will be that

- All variables across $\vec{q}_s, \vec{q}_{s-1}, \ldots, \vec{q}_1$ are still fresh with respect to $\mathfrak{D}_s$, or we are in the case of $s = k$.
- The reification variables for premises and conclusions of $\pi_{s+1}, \pi_{s+2}, \ldots, \pi_k$ are in $\mathfrak{D}_s$.
- The reification variables for all applications of $\pi_{s+1}, \pi_{s+2}, \ldots, \pi_k$ are in $\mathfrak{D}_s$, these are the constraints in equation (5.47).
- The substitution constraints in equation (5.48), for all applications of $\pi_{s+1}, \pi_{s+2}, \ldots, \pi_k$ are in $\mathfrak{D}_s$

Thus, when the step for $s = 1$ is done, we have added all constraints listed in the lemma.

The base case $s = k$*:*
At first we add reifications of all premises of π_k, and the conclusion D_k by proposition 2.11. We let this constraint database be denotes as $\mathfrak{D}_k$. As π_k is not being applied by any other derivation, there are no constraints on the form in 5.47 and 5.48 to add. Notice that all constraints in $\mathfrak{D}_k$ are over variables in $\vec{q}_k$, therefore all variables across $\vec{q}_{k-1}, \vec{q}_{k-2}, \ldots, \vec{q}_1$ are fresh.

Induction step $s < k$*:*
By our induction assumption all variables across $\vec{q}_{s+1}, \vec{q}_{s+2}, \ldots, \vec{q}_k$ are fresh. Specifically because $\vec{q}_s$ is fresh, we can apply proposition 5.3 to add the reification constraints

$$r\left(D_s\restriction_{\lambda_{s,j}}\right) \iff D_s\restriction_{\lambda_{s,j}} \tag{5.49}$$

for all $j \leq J(\pi_s)$ as well as the substitution constraint

$$\bigwedge_{j^*<j} r\left(\mathrm{Imp}_{t_{s,j^*}}(D_s\restriction_{\lambda_{s,j^*}})\right) \wedge \overline{r\left(\mathrm{Imp}_{t_{s,j}}(D_s\restriction_{\lambda_{s,j}})\right)}\,(q_{s,i} \leftrightarrow \lambda_{s,j}(q_{s,i})) \tag{5.50}$$

also for all $j \leq J(\pi_s)$. Lastly we add the reification variables for the premises and conclusion of π_s by proposition 2.11. Know we have added all the constraints for the induction hypothesis, and none of the constraints added include variables from $\vec{q}_{s-1}, \vec{q}_{s-2}, \ldots, \vec{q}_1$.

□

5.2 Simulating in full generality

Now on to the concluding theorem of the thesis. In here we combine the previous lemmas, to show that in full generality, we can translate a proof with lemmas into a VeriPB proof more efficiently than simply inlining all lemma applications. The ideas in this proof are completely similar to the ones already presented, and it purely a matter of arguing that these different ideas can be combined.

Theorem 1. *Let*

$$\pi_1 : P_{1,1}, \dots, P_{1,m_1} \vdash D_1 \tag{5.51}$$
$$\pi_2 : P_{2,2}, \dots, P_{2,m_2} \vdash D_2 \tag{5.52}$$
$$\vdots$$
$$\pi_k : P_{k,1}, \dots, P_{k,m_k} \vdash D_k \tag{5.53}$$

be a sequence of derivations with lemmas, on respectively the variable sets $\vec{q}_1, \vec{q}_2, \dots, \vec{q}_k$ which are all disjoint. Then we can translate this into a VeriPB proof. That is there exists a derivation $\pi^* : P_{k,1}, \dots, P_{k,m_k} \vdash D_k$ which uses only the cutting planes rules of inference along with the redundance-based strengthening rule. Let $M := \max_{s \leq k} m_k$ be the maximum number of premises for any of the derivations, and let $N := \max_{s \leq k} |\vec{q}_s|$ be the maximum number of variables in any of the derivations. If L is accumulated length of $\pi_1, \pi_2, \dots, \pi_k$ then the length of π^* is at most $O(L \cdot MN)$.

Proof. We start by letting $\lambda_{s,j} : \vec{q}_s \to \vec{q}_{t_{s,j}}$ denote the variable substitution for the j'th lemma application of π_s, which is in $\pi_{t_{s,j}}$ for $1 \leq s < t_{s,j} \leq k$. Let $F \doteq \left\{P_{k,1}, P_{k,2}, \dots, P_{k,m_k}\right\}$ be our starting formula. As explained in lemma 5.4, we can reify all premises and conclusions of $\pi_1, \pi_2, \dots, \pi_k$, as well as the conclusions of all lemma applications in $\pi_1, \pi_2, \dots, \pi_k$:

$$\forall s \leq k;\ \ \forall j \leq J(\pi_s)\ :\ r\left(D_s \restriction_{\lambda_{s,j}}\right) \iff D_s \restriction_{\lambda_{s,j}} \tag{5.54}$$

and add all the substitution constraints

$$\bigwedge_{j^* < j} r\left(\mathrm{Imp}_{t_{s,j^*}}(D_s \restriction_{\lambda_{s,j^*}})\right) \wedge \overline{r\left(\mathrm{Imp}_{t_{s,j}}(D_s \restriction_{\lambda_{s,j}})\right)} \implies (q_{s,i} \leftrightarrow \lambda_{s,j}(q_{s,i})) \tag{5.55}$$

for all $s \leq k$, $j \leq J(\pi_s)$, and $q_{s,i} \in \vec{q}_s$. Now we will prove by induction on $t = 1, 2, \dots, k$ that we can derive $\mathrm{Imp}_t(D_t)$ assuming that

- We have already derived $\mathrm{Imp}_s(D_s)$ for all $s < t$.
- For $s < t$, we will define

$$\mathcal{PA}(s,t) := \left\{\mathrm{Imp}_{t_{s,j}}(D_s \restriction_{\lambda_{s,j}}) \;\middle|\; \forall \lambda_{s,j} \text{ where } t_{s,j} < t\right\} \tag{5.56}$$

  be the set of conclusions of all lemma applications of π_s which happen before π_t, i.e. in $\pi_{s+1}, \pi_{s+2}, \dots \pi_{t-1}$. We assume also that all constraints in $\mathcal{PA}(s,t)$ have been derived, for all $s < t$.

Base case $t = 1$:
Deriving $\text{Imp}_1(D_1)$ is done simply by "running" π_1 using
$r\left(P_{1,1}\right) \implies P_{1,1}, r\left(P_{1,2}\right) \implies P_{1,2}, \ldots, r\left(P_{1,m_1}\right) \implies P_{1,m_1}$ as shown in lemma 2.15, because there are no lemma applications in π_1. For the same reason we know that $\mathcal{PA}(1,2) = \emptyset$, and the induction hypothesis is satisfied.

Induction step $t > 1$:
As we have argued many times, any step E in π_t which is not a lemma application we can perform correspondingly to get $\text{Imp}_t(E)$. The main thing to worry about are the lemma applications in π_t. I will only argue for one lemma application that we can simulate it, since it is an argument we have already seen before in the proof of proposition 5.1. So assume want to achieve the conclusion of the lemma application which uses $\lambda_{s,j}$ for some fixed szt and $j \leq J(\pi_s)$, this means that $t_{s,j} = t$. Thereby we also assume that we have already achieved the conclusions of previous applications of π_s in π_t. The substitution constraints for this application are

$$\bigwedge_{j^*<j} r\left(\text{Imp}_{t_{s,j^*}}(D_s\restriction_{\lambda_{s,j^*}})\right) \wedge \overline{r\left(\text{Imp}_t(D_s\restriction_{\lambda_{s,j}})\right)} \implies (q_{s,i} \leftrightarrow \lambda_{s,j}(q_{s,i})) \tag{5.57}$$

Since for all $j^* < j$ we know that the lemma application for λ_{s,j^*} is either in $t_{s,j^*} < t$, where we by induction hypothesis have that $\text{Imp}_{t_{s,j^*}}(D_s\restriction_{\lambda_{s,j^*}})$ has been derived, or it is the case that $t_{s,j^*} = t$ and have been performed already in π_t, and yet again we know that $\text{Imp}_{t_{s,j^*}}(D_s\restriction_{\lambda_{s,j^*}})$ has been derived. By proposition 2.12 we can therefore also derive $r\left(\text{Imp}_{t_{s,j^*}}(D_s\restriction_{\lambda_{s,j^*}})\right) \geq 1$ for all $j^* < j$. Using these we can use proposition 2.13 to remove these reification variables from (5.57) and get

$$\overline{r\left(\text{Imp}_{t_{s,j}}(D_s\restriction_{\lambda_{s,j}})\right)} \implies (q_{s,i} \leftrightarrow \lambda_{s,j}(q_{s,i})) \tag{5.58}$$

From the assumption of the lemma application rule, we also derived

$$\left\{\text{Imp}_t(P_{s,1}\restriction_{\lambda_{s,j}}), \text{Imp}_t(P_{s,2}\restriction_{\lambda_{s,j}}), \ldots, \text{Imp}_t(P_{s,m_s}\restriction_{\lambda_{s,j}})\right\} \tag{5.59}$$

and from these along with $\text{Imp}_s(D_s)$ which we have by our induction assumption, and (5.58) we can use proposition 5.2 to derive $\text{Imp}_t(D_s\restriction_{\lambda_{s,j}})$, as desired. By this argument we can perform all lemma applications in π_t and at the end conclude $\text{Imp}_t(D_t)$. Thereby we thus have also derived all constraints in $\mathcal{PA}(s, t+1)$ for all $s \leq t$.

By the induction above we are able to derive $\text{Imp}_k(D_k)$, and since we have the premises $P_{k,l}$ for $l \leq m_k$, we can use these to get $r\left(P_{k,l}\right) \geq 1$ as explained in proposition 2.12. By knowing that $r\left(P_{k,l}\right) \geq 1$ we can remove them from $\text{Imp}_k(D_k)$ by proposition 2.13 and at the end be able to derive the desired conclusion D_k.

Any cutting planes step in $\pi_1, \pi_2, \ldots, \pi_k$ corresponds to at most $O(M)$ steps in this VeriPB proof, as explained in when performing a derivation on reified premises in lemma 2.15. The steps which are lemma applications do require $O(MN)$ steps in our VeriPB proof, so even if all steps in $\pi_1, \pi_2, \ldots, \pi_k$ were lemma applications the length of the VeriPB proof would be at most $O(L \cdot MN)$. □

6 Conclusion

In this thesis we introduce some of the Pseudo-Boolean reasoning allowed in VeriPB. When doing so, we introduced notation to ease comprehension of later proofs, but thereby also showed that this notation lends itself to a very natural interpretation, through basic cutting planes derivations. We also introduce and discussed what it would mean to extend cutting planes with a rule of inference which allowed reuse of derivations, or as we called it to apply lemmas. The introduction of such a rule does not allow for completely new ways of reasoning, however it does enable proofs to be a polynomial factor shorter, and would thereby be more efficient in practice. How such a rule would be implemented in practice is a project in itself, as there should be some considerations as to how combinatorial solvers work. We also showed that our (perhaps naive) definition for a lemma application rule is incompatible with the redundance-based strengthening rule, and thus incompatible with VeriPB. We therefore showed how derivations using lemmas, could be simulated, or translated into proofs with cutting planes and redundance-based strengthening. We showed this in steps, in which chapter 4 introduced the fundamental idea, that we can add constraints using redundance-based strengthening, which would allow us to not only substitute variables, but also to translate steps which applied lemmas into standard cutting plane proofs. Then it was slowly generalized to handle "lemmas using lemmas" recursively, and finally in theorem 1 we combine all the ideas and show that we in general can translate any sequence of derivations with lemmas into a cutting planes with redundance-based strengthening proof.

6.1 Limitations

As shown in proposition 5.2, to translate a single lemma application step does require the cutting planes derivation to be $O(m \cdot n)$ steps, where m is the number of premises in the lemma and n is the number of variables in the lemma. This means that any translation of a derivation with lemmas can cause an increased of size $O(mn)$ in length. In any case were the length of the lemma being applied is smaller than $O(mn)$ it would be more efficient to instead just inline the lemma, but our belief is that more often than not, the lemma will be significantly longer than $O(mn)$. In practice a factor of $O(mn)$ is significant, and it is not clear how efficiently such a translation could be in practice.

Another limitation of our approach is that it requires that we know the full layout of the derivation using lemmas, before we start the translation. This means we would be unable to translate such a derivation in an online fashion. This also means that even combinatorial solvers could not use this approach to reuse derivations while solving a problem, and we would have to allow them another rule set, and then afterwards translate into a certificate that could be verified by VeriPB.

References

[Ach09] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1:1–41, 2009.

[AGJ+18] Özgür Akgün, Ian P Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Metamorphic testing of constraint solvers. In *Principles and Practice of Constraint Programming: 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings 24*, pages 727–736. Springer, 2018.

[BGMN23] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI '22*.

[BLB10] Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of sat and qbf solvers. In *Theory and Applications of Satisfiability Testing–SAT 2010: 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings 13*, pages 44–57. Springer, 2010.

[BN21] Sam Buss and Jakob Nordström. Proof complexity and sat solving. *Handbook of Satisfiability*, 336:233–350, 2021.

[CCT87] William Cook, Collette R Coullard, and Gy Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987.

[CFHH+17] Luís Cruz-Filipe, Marijn JH Heule, Warren A Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified rat verification. In *Automated Deduction–CADE 26: 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6–11, 2017, Proceedings*, pages 220–236. Springer, 2017.

[CFMSSK17] Luís Cruz-Filipe, Joao Marques-Silva, and Peter Schneider-Kamp. Efficient certified resolution proof checking. In *Tools and Algorithms for the Construction and Analysis of Systems: 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I 23*, pages 118–135. Springer, 2017.

[CKSW13] William Cook, Thorsten Koch, Daniel E Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, 2013.

[DBB17] Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe. Symmetric explanation learning: Effective dynamic symmetry handling for sat. In *Theory and Applications of Satisfiability Testing–SAT 2017: 20th International Conference, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 20*, pages 83–100. Springer, 2017.

[EGM+20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, et al. Justifying all differences using pseudo-boolean reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1486–1494, 2020.

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[GB23] Abhishek Gunjan and Siddhartha Bhattacharyya. A brief review of portfolio optimization techniques. *Artificial Intelligence Review*, 56(5):3847–3886, 2023.

[GMM+20] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 338–357. Springer, 2020.

[GMN21] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 1134–1140, 2021.

[GN03] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for cnf formulas. pages 10886–10891, 01 2003.

[GN21] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.

[Goc22] Stephan Gocht. *Certifying Correctness for Combinatorial Algorithms by Using Pseudo-Boolean Reasoning*. PhD thesis, Lund University, Lund, Sweden, June 2022. Available at `https://portal.research.lu.se/en/publications/certifying-correctness-for-combinatorial-algorithms-by-using-pseu`.

[LBP10] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010.

[MMNS11] Ross M McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

[MO15] David F Manlove and Gregg O'malley. Paired and altruistic kidney donation in the uk: Algorithms and experimentation. *Journal of Experimental Algorithmics (JEA)*, 19:1–21, 2015.

[MSS99] Joao P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[MTW97] Ken McAloon, Carol Tretkoff, and Gerhard Wetzel. Sports league scheduling. In *Proceedings of Third Ilog International Users Meeting*. Citeseer, 1997.

[TD20] Rodrigue Konan Tchinda and Clémentin Tayou Djamegni. On certifying the unsat result of dynamic symmetry-handling-based sat solvers. *Constraints*, 25(3):251–279, 2020.

[Wat95] Michael S Waterman. Applications of combinatorics to molecular biology. *Handbook of combinatorics*, 1:2, 1995.

[WHHJ14] Nathan Wetzler, Marijn JH Heule, and Warren A Hunt Jr. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 422–429. Springer, 2014.

[YQLG16] Lixing Yang, Jianguo Qi, Shukai Li, and Yuan Gao. Collaborative optimization for train scheduling and train stop planning on high-speed railways. *Omega*, 64:57–76, 2016.