Studies in Logic, Vol. 3, No. 3 (2010): 1–23
PII: 1674-3202(2010)-03-0001-23

# The Power of Auxiliary Variables for Propositional and Quantified Boolean Formulas*

Uwe Bubeck
*Computer Science Institute, University of Paderborn*
bubeck@upb.de

Hans Kleine Büning
*Computer Science Institute, University of Paderborn*
kbcsl@upb.de

**Abstract.** Using auxiliary variables to introduce definitions is a popular and powerful technique in knowledge representation which can lead to shorter and more natural encodings without long repetitions. In this paper, we formally define the notion of auxiliary variables and then examine their expressive power and discuss interesting applications.
We relate the idea of reusing intermediate results without copying by definitions to similar concepts in other representations of Boolean functions. In particular, we show that propositional logic with definitions has essentially the same expressive power as Boolean circuits with arbitrary fan-out and existentially quantified Boolean formulas in which the bound variables satisfy the Horn property (called $\exists\text{HORN}^b$).
The paper also considers restrictions on the structure of definitions and extensions of propositional definitions. In particular, we examine the relationship between positive propositional definitions and positive definitions with existential quantification or, equivalently, the relationship between $\exists\text{HORN}^b$ formulas and existentially quantified CNF formulas without the Horn restriction on the bound variables ($\exists\text{CNF}^*$). A further extension is to allow arbitrary quantifiers or, equivalently, nesting of Boolean formulas. The expressive power of the bound variables in a quantified CNF formula is shown to be determined by the structure of minimal unsatisfiable or minimal false subformulas of the bound parts of the clauses.

## 1. Introduction

Driven by the development of increasingly powerful SAT solvers, it has become quite popular in the last few years to solve difficult decision problems by encoding them as propositional formulas. Prominent examples include bounded model checking ([4]) or planning ([11]). The success of SAT-based problem solving relies heavily on finding good propositional encodings of the original problem. This can, however, be a difficult task, because the limited expressive power of propositional logic often prevents the modeling of complex situations in a direct way. Having to use alternate and more indirect formulations is not only time-consuming and error-prone, but also

*Received* 2010-08-03

*Supported by the German Research Foundation (DFG), grant KL 529 / QBF.

increases the risk of ending up with very large encodings, possibly with significant redundancy, especially for interesting real-world problems.

For example, consider the problem of determining whether a set of objects $\{o_1, ..., o_r\}$, given as binary vectors $\{\mathbf{z}_1, ..., \mathbf{z}_r\}$, $\mathbf{z}_i = (z_{i,1}, ..., z_{i,k}) \in \{0,1\}^k$, contains at least one object which satisfies some property $\pi$. If we want to encode this check into a propositional formula which depends exactly on $\mathbf{z}_1, ..., \mathbf{z}_r$, there does not seem to be a feasible alternative to a formula like $\pi(\mathbf{z}_1) \vee ... \vee \pi(\mathbf{z}_r)$. However, having multiple copies of $\pi$ can be very inefficient if $\pi$ itself is a complex formula. The solution is to introduce additional variables which can be used as abbreviations, for example as follows:

$$((\mathbf{x} \leftrightarrow \mathbf{z}_1) \vee ... \vee (\mathbf{x} \leftrightarrow \mathbf{z}_r)) \wedge \pi(\mathbf{x})$$

Again, $\mathbf{x}$ should be a vector of binary variables, and the bi-implications should be understood as being component-wise. The obvious advantage of this encoding is that it requires only one copy of $\pi$. But on the other hand, this new formula is not logically equivalent to $\pi(\mathbf{z}_1) \vee ... \vee \pi(\mathbf{z}_r)$ in the sense that each satisfying assignment of truth values to the variables in one formula must also be a satisfying truth assignment for the other formula. The problem is that this criterion also includes arbitrary values of $\mathbf{x}$, not only the ones satisfying $\pi$. This can be avoided by using a weaker concept of equivalence which considers only the original variables $\mathbf{z}_1, ..., \mathbf{z}_r$ or by augmenting propositional logic with existential quantifiers which bind the additional variables, so that they become local to the formula and are no longer explicitly considered when checking for equivalence. In both approaches, the key element is the distinction between the original variables and newly introduced local variables, which we will in the following refer to as *auxiliary variables*.

Auxiliary variables also play an important role in transforming formulas from one representation into another one. For example, many state-of-the-art SAT solvers are specifically designed for propositional formulas in conjunctive normal form (CNF), that is, conjunctions of disjunctions of positive or negative occurrences of variables. It is well known that there are propositional formulas for which every equivalent CNF formula is exponentially longer. This exponential growth can be avoided by introducing auxiliary variables to abbreviate subformulas which violate the CNF structure. For example, the Tseitin transformation [16, 19] produces linear-size CNF formulas by successively replacing non-CNF subformulas of the form $\alpha \vee (\beta \wedge \pi)$ with $(\alpha \vee x) \wedge (x \rightarrow (\beta \wedge \pi))$ or, equivalently, $(\alpha \vee x) \wedge (\neg x \vee \beta) \wedge (\neg x \vee \pi)$ for a new auxiliary variable $x$.

The examples presented so far illustrate that auxiliary variables are an important tool for obtaining short propositional encodings. The main goal of this work is to explore the expressive power of auxiliary variables in more detail. We will see that the idea of using auxiliary variables to define abbreviations for reusing intermediate results without copying can be related to similar concepts in other representations of

Boolean functions, for example the fan-out in Boolean circuits. Along the way, we also present typical encoding patterns for modeling with auxiliary variables, which can be very helpful when developing new encodings or transformations.

## 2. Preliminaries

We begin by recalling the basic concepts and terminology which will be needed in the following sections.

It is often useful to assume that propositional formulas are given in a canonical normal form. *Negation normal form* (NNF) requires that every negation occurs immediately in front of a variable. A *literal* is then a positive propositional variable ($v$) or a negated variable ($\neg v$), and a *clause* $C = l_1 \vee ... \vee l_h$ is a disjunction of literals. As mentioned in the introduction, a formula is in *conjunctive normal form* (CNF) if and only if it is a conjunction of clauses $\phi = C_1 \wedge ... \wedge C_q$. The dual *disjunctive normal form* (DNF) denotes a disjunction of conjunctions of literals.

Quantified Boolean formulas (QBF) extend propositional logic with quantifiers over variables. $\forall x\ \phi(x)$ is defined to be true if and only if $\phi(0)$ is true *and* $\phi(1)$ is true. Variables which are bound by universal quantifiers are called *universal variables*. Similarly, $\exists y\ \phi(y)$ is defined to be true if and only if $\phi(0)$ *or* $\phi(1)$ is true. In this case, $y$ is called an *existential variable.* To save parentheses, we assume that the logical connectives have a higher binding priority than the quantifiers. A quantified Boolean formula $\Phi$ is in *prenex form* if it has the form $\Phi = Q_1v_1...Q_kv_k\,\phi$ with quantifiers $Q_i \in \{\forall, \exists\}$ and a propositional formula $\phi$. We call $Q := Q_1v_1...Q_kv_k$ the *prefix* and $\phi$ the *matrix* of $\Phi$. Unless mentioned otherwise, we assume that QBF formulas are always in prenex form.

Variables which are not bound by quantifiers are *free* variables. Formulas without free variables are called *closed.* If free variables are allowed, we indicate this with an additional star * after the name of the formula class. Accordingly, QBF is the class of closed quantified Boolean formulas, and QBF* denotes quantified Boolean formulas with free variables.

A closed QBF formula is either *true* or *false*. It is true if and only if there exists an assignment of truth values to the existential variables depending on the preceding universal variables such that the propositional matrix of the formula is true for all values of the universal variables. For example, $\Phi = \forall x \exists y\ (x \vee y) \wedge (\neg x \vee \neg y)$ is true, because by choosing $y = 1$ when $x = 0$ and $y = 0$ when $x = 1$, we can satisfy the formula for all values of $x$.

The truth value of a QBF* formula depends on the values of the free variables. A QBF* formula $\Phi(\mathbf{z})$ is *satisfiable* if and only if there exists a truth assignment $t(\mathbf{z}) := (t(z_1), ..., t(z_r)) \in \{0,1\}^r$ to the free variables $\mathbf{z} = (z_1, ..., z_r)$ for which $\Phi(t(\mathbf{z}))$ is true. Here, $\Phi(t(\mathbf{z}))$ denotes the closed QBF formula that results from substituting the truth values $t(z_1), ..., t(z_r)$ for the free variables $z_1, ..., z_r$. For example, the formula

$\Phi(z) = \forall x \exists y\ (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y) \wedge (\neg y \vee z)$ is satisfiable: for $z = 1$, we can choose $y = \neg x$, and substituting $z$ and $y$ produces the tautological matrix $(x \vee \neg x \vee 0) \wedge (\neg x \vee x) \wedge (x \vee 1)$. In that particular example, the formula is also true for $z = 0$: in this case, we can choose $y = 0$, and all clauses are satisfied.

It is well known that determining the satisfiability of formulas in QBF or QBF* is a PSPACE-complete problem [15], which is assumed to be significantly harder than the NP-completeness of the propositional SAT problem. More precisely, it appears that there is a close relationship between the number of quantifier alternations in the prefix of quantified Boolean formulas and the complexity of the corresponding satisfiability problem, giving rise to the *polynomial-time hierarchy* of complexity classes [18, 21]. It is clear that purely existentially quantified Boolean formulas (∃BF* or ∃CNF* if the matrix is in CNF) are not more difficult to solve than propositional formulas, because in this case, it is sufficient to find a satisfying truth assignment to the matrix. An alternative to placing restrictions on the prefix is to consider quantified Boolean formulas with restrictions on the structure of the matrix. Well-known tractable examples are quantified Horn formulas (QHORN*) [8] and quantified 2-CNF (Q2-CNF*) [2].

In this paper, our main interest is to compare the expressive power of different formula classes. This involves determining for formulas in one class the length of the shortest equivalent formula in the other class. Two QBF* formulas $\Phi_1(z_1, ..., z_r)$ and $\Phi_2(z_1, ..., z_r)$ are said to be *equivalent* ($\Phi_1 \approx \Phi_2$) if and only if $\Phi_1 \models \Phi_2$ and $\Phi_2 \models \Phi_1$, where *semantic entailment* $\models$ is defined as follows: $\Phi_1 \models \Phi_2$ if and only if for all truth assignments $t(\mathbf{z}) = (t(z_1), ..., t(z_r)) \in \{0, 1\}^r$ to the free variables $\mathbf{z} = (z_1, ..., z_r)$, we have $\Phi_1(t(\mathbf{z})) = 1 \Rightarrow \Phi_2(t(\mathbf{z})) = 1$. That means $\Phi_1 \approx \Phi_2$ if and only if $\Phi_1(t(\mathbf{z})) = \Phi_2(t(\mathbf{z}))$ for all assignments $t(\mathbf{z}) \in \{0, 1\}^r$. These definitions apply to propositional formulas as well if we treat them as QBF* formulas without quantifiers. That also allows us to consider equivalence between a QBF* formula on the one hand and a propositional formula on the other hand. In this case, the free variables of the QBF* formula correspond to the variables in the propositional formula, and for all truth value assignments to them, both formulas must evaluate to the same truth value.

The notion of equivalence can also be extended to different representations of Boolean functions which are not formulas, e.g. Boolean circuits. A *Boolean circuit* is an acyclic directed graph with exactly one outgoing edge and some input nodes that are labeled with Boolean variables. The other nodes are AND-, OR-, and NOT-gates that each have two (AND and OR) or one (NOT) incoming and an arbitrary number of outgoing edges. The *fan-out* of a circuit is the maximum number of outgoing edges of the AND- and OR-gates. By De Morgan's law, we can transform in linear time an arbitrary circuit into an equivalent circuit in *standard form*, where the inner nodes are only AND- and OR-gates and the inputs are variables $x$ and/or negated variables $\neg x$. Subsequently, when we talk about circuits, we always mean circuits in standard form. The output of a circuit $c$ over variables $z_1, ..., z_r$, each occurring positively and/or negatively as input, is written as $c(z_1, ..., z_r)$. We say that $c$ is equivalent to a QBF* or propositional formula $\Phi$ with (free) variables $z_1, ..., z_r$ if and only if $c(t(z_1), ..., t(z_r)) = \Phi(t(z_1), ..., t(z_r))$ for all truth assignments $t(\mathbf{z}) = (t(z_1), ..., t(z_r)) \in \{0, 1\}^r$. In terms of Boolean functions, $c \approx \Phi$ means that $c$ and $\Phi$ represent the same Boolean function.

For a propositional or quantified Boolean formula $\Phi$, we let $|\Phi|$ denote the length of $\Phi$. It is counted as the number of occurrences of variables, including the quantified variables in the prefix in the case of a quantified Boolean formula. For example, the formula $\forall x \exists y \ (x \vee y) \wedge (\neg x \vee \neg y)$ has length 6. For a circuit $C$, we let the circuit size $|C|$ be the number of gates in $C$.

## 3. Auxiliary Variables

In this section, we will introduce a formal definition of auxiliary variables. It is motivated by the following initial examples.

Let $\alpha = (a \vee b \vee c \vee d)$ be a clause with four literals. When we want to represent this clause by 3-clauses, we introduce a new variable $x$ and replace $\alpha$ with $\beta = (a \vee b \vee x) \wedge (\neg x \vee c \vee d)$. Clearly, $\alpha$ and $\beta$ are not equivalent. But with respect to formulas $\sigma$ over the variables $a, b, c, d$, it holds that $\alpha \models \sigma$ iff $\beta \models \sigma$.

As mentioned before, the Tseitin procedure for linear-size CNF transformation replaces subformulas of the form $\alpha \vee (\beta \wedge \pi)$ with $(\alpha \vee x) \wedge (x \rightarrow (\beta \wedge \pi)) \approx (\alpha \vee x) \wedge (\neg x \vee \beta) \wedge (\neg x \vee \pi)$ for a new variable $x$. Again, the original and the resulting formula have the same consequences over the variables occurring in $\alpha, \beta$ and $\pi$.

The last example shows a length reduction by auxiliary variables. For the formula $\phi = \bigwedge_{1 \leq i,j \leq n} (a_i \vee b_j)$, we choose $\varphi = \bigwedge_{1 \leq i \leq n} (a_i \vee x) \wedge \bigwedge_{1 \leq j \leq n} (\neg x \vee b_j)$. Then both formulas are again equivalent with respect to the original variables. This kind of restricted equivalence is formally introduced in the following definition.

**Definition 1** (Restricted Equivalence) [9, 13]

Let $\mathbf{z} = z_1, ..., z_n$ be variables, and let $\alpha$ and $\beta$ be propositional formulas. $\alpha$ is **restricted equivalent** to $\beta$ for $\mathbf{z}$ if and only if for every propositional formula $\sigma$ over the variables $\mathbf{z}$, it holds that $\alpha \models \sigma$ iff $\beta \models \sigma$. For the restricted equivalence, we write $\alpha \approx_{\{\mathbf{z}\}} \beta$.

For propositional Horn formulas, the equivalence problem can be solved in quadratic time, since the satisfiability problem is decidable in linear time. ([13]) But the restricted equivalence problem is coNP-complete for Horn formulas. ([9])

We have just seen how to transform clauses with more than three literals into 3-clauses. Can we apply a similar idea to shorten a conjunctive term $\alpha = (a \wedge$

$b \wedge c \wedge d)$ inside a formula in disjunctive normal form? If we replace $\alpha$ with $\beta = (a \wedge b \wedge x) \vee (\neg x \wedge c \wedge d)$, both formulas are only equivalent with respect to $a$, $b$, $c$ and $d$ if we require that $\beta$ holds for all possible values of $x$, that means if we bind $x$ with a universal quantifier. We end up with a quantified Boolean formula $\forall x\ (a \wedge b \wedge x) \vee (\neg x \wedge c \wedge d)$. Similar to the introductory examples at the beginning of this section, $x$ is used only as a helper variable which has no meaning in the original formula $\alpha$. Accordingly, the formal definition of auxiliary variables should reflect not only the situation in the earlier examples, but also the last example with a universally quantified variable. By negating $\alpha$ and $\beta$, this last example can be mapped to the dual problem of shortening long CNF clauses, for which we have observed that $\alpha \approx_{\{\mathbf{z}\}} \beta$. That means $\neg\alpha \approx_{\{\mathbf{z}\}} \neg\beta$ characterizes the last example, so we also include this case into the following definition of auxiliary variables. Please notice that in general, $\alpha \approx_{\{\mathbf{z}\}} \beta$ does not imply $\neg\alpha \approx_{\{\mathbf{z}\}} \neg\beta$. For example, for the formulas $\phi = a$ and $\psi = a \wedge x$, we have $\phi \approx_{\{a\}} \psi$, but not $\neg a \approx_{\{a\}} \neg a \vee \neg x$, since $\neg a \vee \neg x \not\models \neg a$.

**Definition 2** (Auxiliary Variables)

For variables $\mathbf{z} = z_1, ..., z_n$ and $\mathbf{x} = x_1, ..., x_m$, let $\alpha(\mathbf{z})$ and $\beta(\mathbf{z}, \mathbf{x})$ be formulas over the variables $\mathbf{z}$ and $\mathbf{z}, \mathbf{x}$. The variables $\mathbf{x}$ are called **auxiliary variables** for $\alpha(\mathbf{z})$ and $\beta(\mathbf{z}, \mathbf{x})$ if and only if $\alpha(\mathbf{z}) \approx_{\{\mathbf{z}\}} \beta(\mathbf{z}, \mathbf{x})$ or $\neg\alpha(\mathbf{z}) \approx_{\{\mathbf{z}\}} \neg\beta(\mathbf{z}, \mathbf{x})$.

The definition of auxiliary variables by restricted equivalence for the free variables $\mathbf{z}$ can be replaced with full logical equivalence when adding quantifiers for the auxiliary variables.

**Lemma 3** (Alternative Description of Auxiliary Variables)

Let $\alpha(\mathbf{z})$ and $\beta(\mathbf{z}, \mathbf{x})$ be formulas over the variables $\mathbf{z}$ and $\mathbf{z}, \mathbf{x}$. Then the following holds:

1. $\alpha(\mathbf{z}) \approx_{\{\mathbf{z}\}} \beta(\mathbf{z}, \mathbf{x})$ if and only if $\alpha(\mathbf{z}) \approx \exists\mathbf{x}\beta(\mathbf{z}, \mathbf{x})$.
2. $\neg\alpha(\mathbf{z}) \approx_{\{\mathbf{z}\}} \neg\beta(\mathbf{z}, \mathbf{x})$ if and only if $\alpha(\mathbf{z}) \approx \forall\mathbf{x}\beta(\mathbf{z}, \mathbf{x})$.
3. The variables $\mathbf{x}$ are auxiliary variables for $\alpha(\mathbf{z})$ and $\beta(\mathbf{z}, \mathbf{x})$ if and only if $\alpha(\mathbf{z}) \approx \exists\mathbf{x}\beta(\mathbf{z}, \mathbf{x})$ or $\alpha(\mathbf{z}) \approx \forall\mathbf{x}\beta(\mathbf{z}, \mathbf{x})$.

**Proof** 1. From left to right, $\alpha(\mathbf{z}) \approx_{\{\mathbf{z}\}} \beta(\mathbf{z}, \mathbf{x})$ implies that $\beta(\mathbf{z}, \mathbf{x}) \models \alpha(\mathbf{z})$ and $\alpha(\mathbf{z}) \models \beta(\mathbf{z}, 0, ..., 0, 0) \vee \beta(\mathbf{z}, 0, ..., 0, 1) \vee ... \vee \beta(\mathbf{z}, 1, .., 1, 0) \vee \beta(\mathbf{z}, 1, .., 1, 1)$. The latter is equivalent to $\alpha(\mathbf{z}) \models \exists\mathbf{x}\beta(\mathbf{z}, \mathbf{x})$, and $\beta(\mathbf{z}, \mathbf{x}) \models \alpha(\mathbf{z})$ implies $\exists\mathbf{x}\beta(\mathbf{z}, \mathbf{x}) \models \alpha(\mathbf{z})$. In total, $\alpha(\mathbf{z}) \approx \exists\mathbf{x}\beta(\mathbf{z}, \mathbf{x})$.
From right to left, $\alpha(\mathbf{z}) \approx \exists\mathbf{x}\beta(\mathbf{z}, \mathbf{x})$ implies that $\alpha(\mathbf{z}) \models \sigma(\mathbf{z})$ for some $\sigma(\mathbf{z})$ if and only if $\exists\mathbf{x}\beta(\mathbf{z}, \mathbf{x}) \models \sigma(\mathbf{z})$, which in turn holds if and only if $\beta(\mathbf{z}, \mathbf{x}) \models \sigma(\mathbf{z})$.
2. $\alpha(\mathbf{z}) \approx \forall\mathbf{x}\beta(\mathbf{z}, \mathbf{x})$ if and only if $\neg\alpha(\mathbf{z}) \approx \neg(\forall\mathbf{x}\beta(\mathbf{z}, \mathbf{x}))$. With the quantifier inversion rule $\neg(\forall\mathbf{x}\beta(\mathbf{z}, \mathbf{x})) \approx \exists\mathbf{x}\neg\beta(\mathbf{z}, \mathbf{x})$, the claim follows immediately

from (1).

3. This is the result of applying (1) and (2) to Definition 2.

□

When working with formulas in CNF, it is a good idea to separate literals over auxiliary variables in a clause from literals over original variables. Then we can attempt to characterize the expressive power of auxiliary variables depending on the structure of the auxiliary parts of clauses. On the basis of the previous lemma, we can also consider quantified formulas with literals over bound variables on the one hand and literals over free variables on the other hand. For a quantified Boolean formula $\Phi = Q_1 v_1 ... Q_n v_n \ \phi_1 \wedge ... \wedge \phi_q$, $Q_i \in \{\forall, \exists\}$, we write $\phi_k = \phi_k^b \vee \phi_k^f$ where the *bound part* $\phi_k^b$ contains all literals in the clause which are over a quantified variable $v_i$, and the *free part* $\phi_k^f$ contains all free literals.

We will now show that the structure of minimal unsatisfiable or minimal false subformulas of the bound parts has a strong influence on the expressive power of the bound or auxiliary variables. A CNF formula $\phi = \phi_1 \wedge ... \wedge \phi_q$ is called *minimal unsatisfiable* if and only if $\phi$ is unsatisfiable and the removal of an arbitrary clause $\phi_i$ produces a satisfiable formula. A quantified Boolean formula $\Psi = Q \bigwedge_{1 \leq i \leq q} \psi_i$ with CNF matrix and without free variables is called *minimal false* if and only if $\Psi$ is false and removing an arbitrary clause $\psi_i$ leads to a true formula. If $\Psi$ is purely existentially quantified, it is minimal false if and only if the matrix is minimal unsatisfiable.

Let $\exists \mathbf{x} \ \phi^b = \exists x_1 ... \exists x_n \ \phi_1^b \wedge ... \wedge \phi_q^b$ be a minimal false formula over the variables $\mathbf{x} = x_1, ..., x_n$, and let $\phi_1^f, ..., \phi_q^f$ be formulas over a different set of variables $\mathbf{z}$. Then it holds that $\exists x_1 ... \exists x_n \ (\phi_1^b \vee \phi_1^f) \wedge ... \wedge (\phi_q^b \vee \phi_q^f)$ is equivalent to $\phi_1^f \vee ... \vee \phi_q^f$. The reason is that making at least one of the free parts true has the same effect as removing at least one of the bound parts, which makes the remainder true due to the minimal falsity of $\exists \mathbf{x} \ \phi^b$. The same idea applies if we also allow universal quantifiers and $Q_1 v_1 ... Q_n v_n \ \phi^b$ is minimal false.

In general, the quantified bound parts of a quantified CNF formula are not necessarily minimal false, but they should at least be false. Otherwise, the formula would be true regardless of the free parts and therefore be tautological with respect to the free variables. With the quantified bound parts being false, we can lift the previous idea by considering all their minimal false subformulas. The following example illustrates this. Let

$$\Phi = \forall x \exists y_0 \exists y_1 \ (y_0 \vee z_0) \wedge (\neg y_0 \vee z_1) \wedge (\neg y_0 \vee z_2) \wedge (y_1 \vee z_3) \wedge (x \vee z_4)$$

be a formula with quantified variables $x, y_0, y_1$ and free variables $z_0, ..., z_4$. Then the quantified bound parts

$$\forall x \exists y_0 \exists y_1 \ \phi_1^b \wedge \phi_2^b \wedge \phi_3^b \wedge \phi_4^b \wedge \phi_5^b = \forall x \exists y_0 \exists y_1 \ y_0 \wedge \neg y_0 \wedge \neg y_0 \wedge y_1 \wedge x$$

are false and contain 3 minimal false subformulas: $\exists y_0\ \phi_1^b \wedge \phi_2^b = \exists y_0\ y_0 \wedge \neg y_0$, $\exists y_0\ \phi_1^b \wedge \phi_3^b = \exists y_0\ y_0 \wedge \neg y_0$ and $\forall x\ \phi_5^b = \forall x\ x$. Notice that $\exists y_0\ y_0 \wedge \neg y_0$ occurs twice: once for $\phi_1^b \wedge \phi_2^b$ and once for $\phi_1^b \wedge \phi_3^b$. The original formula $\Phi$ can only become satisfied if each of these minimal false subformulas is disabled by satisfying at least one of the corresponding free parts. That means $\phi_1^f \vee \phi_2^f$ must be satisfied, as well as $\phi_1^f \vee \phi_3^f$, and also $\phi_5^f$. It follows that $\Phi \approx (\phi_1^f \vee \phi_2^f) \wedge (\phi_1^f \vee \phi_3^f) \wedge \phi_5^f$. In general, we have the following theorem:

**Lemma 4** (MF Skeleton)

Let $\Phi = Q \bigwedge_{1 \le i \le q} (\phi_i^b \vee \phi_i^f)$ be a quantified CNF formula with non-empty bound parts $\phi_i^b$ and free parts $\phi_i^f$. Let

$$S(\Phi) := \left\{ \Phi' \mid \Phi' = Q\phi_{i_1}^b \wedge ... \wedge \phi_{i_r}^b \text{ is minimal false}, 1 \le i_1, ..., i_r \le q \right\}$$

be the set of minimal false subformulas of the quantified bound parts of $\Phi$. Then the following equivalence holds:

$$\Phi \approx \bigwedge_{(Q\phi_{i_1}^b \wedge ... \wedge \phi_{i_r}^b) \in S(\Phi)} (\phi_{i_1}^f \vee \ldots \vee \phi_{i_r}^f)$$

**Proof** Let $M(\Phi) := \bigwedge_{(Q\phi_{i_1}^b \wedge ... \wedge \phi_{i_r}^b) \in S(\Phi)} (\phi_{i_1}^f \vee \ldots \vee \phi_{i_r}^f)$ be the right side of the equivalence. From right to left, let $M(\Phi)$ be true for a truth assignment $\tau$ to the free variables. Suppose $\tau(\Phi)$ is false. Let $Q\phi' := Q(\phi_{i_1}^b \wedge \ldots \wedge \phi_{i_r}^b)$ be the quantified bound parts for which $\tau(\phi_{i_k}^f)$ is false for $1 \le k \le r$. Under the assumption that $\tau(\Phi)$ is false, $Q\phi'$ is also false and contains therefore a minimal false subformula, say $Q\phi^* := Q(\phi_{j_1}^b \wedge \ldots \wedge \phi_{j_t}^b)$. Since $\tau(M(\Phi))$ is true, one of the free parts $\phi_{j_1}^f, ..., \phi_{j_t}^f$ must be true for $\tau$. That is a contradiction.
From left to right, let $\Phi$ be true for a truth assignment $\tau$ to the free variables. Suppose $\tau(M(\Phi))$ is false. Then there is a clause $\phi' := (\phi_{i_1}^f \vee \ldots \vee \phi_{i_r}^f)$ in $M(\Phi)$ for which $\tau(\phi_{i_k}^f)$ is false for $1 \le k \le r$. Since $Q(\phi_{i_1}^b \wedge \ldots \wedge \phi_{i_r}^b)$ is minimal false, we can conclude that $\tau(\Phi)$ is false in contradiction to our assumption. □

Deciding whether or not a formula is minimal false is usually a difficult task and at least as hard as the satisfiability problem. Another problem is that a formula $Q_1 v_1 ... Q_n v_n\ \phi^b$ might have exponentially many minimal false subformulas. This holds even for relatively simple subclasses of quantified Boolean formulas. For instance, it is well known that propositional Horn formulas can have exponentially many minimal unsatisfiable subformulas.

## 4. Propositional Definitions

Definitions play a very important role in knowledge representation, because they do not only lead to shorter encodings avoiding long repetitions, but they can also contribute to a better understanding and to short and easy proofs.

A popular and powerful kind of definition is based on introducing new names for propositional formulas. For example, let $x_1 := a \wedge b$, $x_2 := c \vee \neg x_1$, and $x_3 := (x_1 \wedge \neg a) \vee (c \wedge x_2)$. That means $x_1$ equals the formula $a \wedge b$, and for $x_2$, we obtain $c \vee \neg(a \wedge b)$ after substituting $a \wedge b$ for the occurrence of $x_1$. Finally, $x_3$ abbreviates the formula $((a \wedge b) \wedge \neg a) \vee (c \wedge (c \vee \neg(a \wedge b)))$. When such a sequence of nested abbreviations $x_i$ is unfolded by substituting propositional formulas step by step, we may end up with a propositional formula of exponential length. But we will later see that every Boolean circuit can be represented by a system of definitions of linear length, and vice versa. That illustrates the expressive power of these definitions, in addition to their advantage of more natural formulations. However, expressions of the form $x_i := \phi_i$ are not propositional or quantified Boolean formulas and require an extension of the syntax. So, most current SAT-solvers cannot deal directly with such definitions of abbreviations. A well-known alternative is to use bi-implications $y \leftrightarrow \alpha$, where $\alpha$ is a propositional formula and $y$ is a new variable. Since we want to consider also compound definitions that consist of multiple such bi-implications, we introduce the following notation:

**Definition 5** (Propositional Definition)

Let $\alpha_1, ..., \alpha_m$ be propositional formulas. Then a sequence of bi-implications $D(y_1, ..., y_m) = (y_1 \leftrightarrow \alpha_1, ..., y_m \leftrightarrow \alpha_m) := (y_1 \leftrightarrow \alpha_1) \wedge ... \wedge (y_m \leftrightarrow \alpha_m)$ is called a **definition** if and only if $var(\alpha_i) \cap \{y_i, ..., y_m\} = \emptyset$ for $i = 1, ..., m$.

The introduced variables $y_i$ are called **def-variables**, and the other variables $\cup_i\, var(\alpha_i) \setminus \{y_1, ..., y_m\}$ are called **original variables**.

Every definition $D(y_1, ..., y_m)$ specifies a sequence of **defined formulas**

$$def_D(y_1) := \alpha_1 \text{ and } def_D(y_i) := \alpha_i[y_{i-1}/\alpha_{i-1}]...[y_1/\alpha_1] \text{ for } i = 2, ..., m$$

where $\beta[y/\sigma]$ is the result of substituting $\sigma$ for every occurrence of $y$ in $\beta$.

We define the **length** of $D$ as $|D(y_1, ..., y_m)| := |(y_1 \leftrightarrow \alpha_1) \wedge ... \wedge (y_m \leftrightarrow \alpha_m)|$.

The constraint $var(\alpha_i) \cap \{y_i, ..., y_m\} = \emptyset$ is necessary in order to avoid cyclic dependencies like $y_i \leftrightarrow y_j, y_j \leftrightarrow y_i$.

We can safely use a def-variable as an abbreviation of the defined formula, because an occurrence of a def-variable in combination with its definition has the same consequences over the original variables as the corresponding defined formula:

**Lemma 6** (Definitions and Auxiliary Variables)

Let $D(y_1, ..., y_m) = (y_1 \leftrightarrow \alpha_1, ..., y_m \leftrightarrow \alpha_m)$ be a definition.

For $i = 1, ..., m$, it holds that

$$\exists y_1 ... \exists y_m\ (D(y_1, ..., y_m) \wedge y_i) \approx def_D(y_i)$$

and the def-variables $y_1, ..., y_m$ are auxiliary variables for $D(y_1, ..., y_m) \wedge y_i$ and $def_D(y_i)$.

**Proof**

$$\begin{aligned} & \exists y_1 ... \exists y_m \ (D(y_1, ..., y_m) \wedge y_i) \\ \approx \ & \exists y_1 ... \exists y_{i-1} \exists y_{i+1} ... \exists y_m \left( \bigwedge_{j \neq i} (y_j \leftrightarrow \alpha_j) \wedge \alpha_i \right) \\ \approx \ & \exists y_1 ... \exists y_{i-1} \exists y_{i+1} ... \exists y_m \left( \bigwedge_{j \neq i} (y_j \leftrightarrow \alpha_j) \wedge \alpha_i [y_{i-1}/\alpha_{i-1}] ... [y_1/\alpha_1] \right) \\ \approx \ & \left( \exists y_1 ... \exists y_{i-1} \exists y_{i+1} ... \exists y_m \bigwedge_{j \neq i} (y_j \leftrightarrow \alpha_j) \right) \wedge \alpha_i [y_{i-1}/\alpha_{i-1}] ... [y_1/\alpha_1] \end{aligned}$$

Now, it is easy to see that $\exists y_1 ... \exists y_{i-1} \exists y_{i+1} ... \exists y_m \bigwedge_{j \neq i} (y_j \leftrightarrow \alpha_j)$ is a tautology. With $\alpha_i [y_{i-1}/\alpha_{i-1}] ... [y_1/\alpha_1] = def_D(y_i)$, we immediately have the equivalence.

Then it follows from Lemma 3 that $y_1, ..., y_m$ are auxiliary variables. □

For practical applications, it might be even more useful to have an equivalence like $\exists y_1 ... \exists y_m \ (D(y_1, ..., y_m) \wedge \phi) \approx \phi[y_m/def_D(y_m)] ... [y_1/def_D(y_1)]$, where $\phi$ is an arbitrary formula in which def-variables are used, and the equivalence makes sure that this formula behaves just like the original formula without abbreviations. Such an equivalence can easily be obtained from Lemma 6:

**Corollary 7** $\exists y_1 ... \exists y_m \ (D(y_1, ..., y_m) \wedge \phi) \approx \phi[y_m/def_D(y_m)] ... [y_1/def_D(y_1)]$ for a definition $D(y_1, ..., y_m) = (y_1 \leftrightarrow \alpha_1, ..., y_m \leftrightarrow \alpha_m)$ and an arbitrary propositional formula $\phi$.

**Proof** Introduce an additional variable $y_{m+1}$ with the definition $y_{m+1} \leftrightarrow \phi$ and apply Lemma 6 for $i = m + 1$. □

Definitions have been applied successfully in finding shorter proofs. For example, extended resolution [19] combines the ordinary resolution rule [17] with an *extension rule* that allows the introduction of definitions. For the resolution calculus, it has been shown [10] that the pigeon-hole formulas require exponentially many resolution steps. The standard representation of the pigeon-hole principle encodes that there is no one-to-one mapping between two sets having $n + 1$ and $n$ elements. This leads to the following formula:

For $1 \leq i \leq n$ and $1 \leq j \leq n + 1$, we have variables $x_{i,j}$, where the index $i$ denotes the holes, $j$ denotes the pigeons, and $x_{i,j}$ has the intended meaning "pigeon $j$ is in hole $i$". The resulting formula $\varphi_n = \alpha_n \wedge \beta_n$ contains two kinds of constraints:

1. In every hole there is at most one pigeon:
   $\alpha_n = \bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq k < j \leq n+1} (\neg x_{i,k} \vee \neg x_{i,j})$
2. Every pigeon is in one hole:
   $\beta_n = \bigwedge_{1 \leq j \leq n+1} (x_{1,j} \vee ... \vee x_{n,j})$

There is some constant $c > 1$ for which every resolution refutation of $\varphi_n$ requires at least $c^n$ resolution steps. For the extended resolution, however, it is well-known that a proof can be constructed in $O(n^4)$ steps [6]. It is an open question whether for any formula in conjunctive normal form there is a short proof by extended resolution. But unless NP=coNP, there must exist formulas for which every proof requires superpolynomially many steps.

The power of definitions can also be seen when relating them to Boolean circuits. According to the following lemma, propositional definitions can compactly encode circuits, and vice versa.

**Lemma 8** (Expressive Power of Propositional Definitions)
1. There exists a polynomial $p$, such that for every definition $D(y_1, ..., y_m)$ there are circuits $c_1, ..., c_m$ equivalent to $def_D(y_1), ..., def_D(y_m)$ and the size of $c_i$ is less than $p(|D|)$.
2. For every circuit $c$, there exists a definition $D(y_1, ..., y_m)$, such that $c$ is equivalent to $def_D(y_m)$ and the length of $D$ is linear in the size of $c$.

**Proof** 1. $def_D(y_1) = \alpha_1$ is a propositional formula, so it is clear that it can be represented by an equivalent circuit $c_1$ of polynomial size.
Similarly, for $\alpha_2$ there is a poly-size circuit $c_{\alpha_2}$ in standard form that has inputs labeled with positive and negated instances of the variables $var(\alpha_2)$. We can now combine $c_1$ and $c_{\alpha_2}$ into a poly-size circuit $c_2$ equivalent to $def_D(y_2) = \alpha_2[y_1/\alpha_1]$ by connecting the output of $c_1$ to the input $y_1$ of $c_{\alpha_2}$ and the negated output of $c_1$ to the input $\neg y_1$ of $c_{\alpha_2}$.
For $i = 3, ..., m$, we can analogously obtain $c_i$ from a circuit $c_{\alpha_i}$ for $\alpha_i$ into which the outputs of $c_1, ..., c_{i-1}$ are fed. Notice that $c_{i-1}$ might itself rely on the outputs of $c_1, ..., c_{i-2}$, and $c_{i-2}$ on $c_1, ..., c_{i-3}$, and so on, which would cause the size of $c_i$ to be exponential. But we can make use of fan-out and simultaneously feed the output of a circuit $c_j$, $j = 1, ..., i-1$, into all subsequent circuits $c_{j+1}, ..., c_i$. That means each such circuit $c_j$ needs to occur at most once in $c_i$, and it follows that the size of $c_i$ remains polynomial.

2. Let $v_{i_1}, ..., v_{i_r}$ be a topological ordering of the gates in $c$. Then we associate with every gate $v_{i_j}$ a bi-implication $y_{i_j} \leftrightarrow (u_1 \circ u_2)$, with $\circ = \wedge$ if $v_{i_j}$ is an AND gate and $\circ = \vee$ for an OR gate. $y_{i_j}$ is a new variable, and $u_1, u_2 \in var(c) \cup \{y_{i_1}, ..., y_{i_{j-1}}\}$ are the inputs to the gate, where outputs of preceding gates are represented by previously introduced def-variables. Obviously, the sequence of all bi-implications exactly describes the circuit $c$, and its size is linear in the number of gates of $c$.

□

## 5. Restrictions on the Structure of Definitions

Now, we will discuss the expressive power of propositional definitions under various constraints. Let $D(y_1, ..., y_m) = (y_1 \leftrightarrow \alpha_1, ..., y_m \leftrightarrow \alpha_m)$, then a natural type of constraint is to enforce restrictions on the occurrences of literals over def-variables $y_1, ..., y_{i-1}$ in each formula $\alpha_i$. When we restrict the number of such literals, we can distinguish the following three cases:

1. In the easiest case, the formulas $\alpha_1, ..., \alpha_m$ do not contain any literals over def-variables, which we call a *pure* definition. For pure definitions, it is clear that $def_D(y_i) = \alpha_i$, which means the expressive power is the same as that of propositional formulas.
2. If each formula $\alpha_i$ may contain at most one literal over a def-variable, the definition is called *simple*. Since $def_D(y_i) = \alpha_i[y_{i-1}/\alpha_{i-1}]...[y_1/\alpha_1]$ and every def-variable occurs at most once, the length of $def_D(y_i)$ cannot exceed the length of $D$. That means pure definitions and simple definitions have the same expressive power.
3. In the case of arbitrary bi-implications $y_i \leftrightarrow \alpha_i$, we can restrict ourselves to bi-implications of the form $y \leftrightarrow L_1 \vee L_2$ where $L_j$ is a literal. Let $y \leftrightarrow \alpha \vee \beta$ where $\alpha$ is not a literal, then we can replace this by two bi-implications $y \leftrightarrow y_1 \vee \beta, y_1 \leftrightarrow \alpha$, and analogously for $\beta$. We can assume that $\alpha$ ($\beta$, resp.) is in negation normal form. If $\alpha$ ($\beta$, resp.) is itself a disjunction, we apply the previous rule again. Otherwise, a bi-implication with a conjunction $x \leftrightarrow \phi \wedge \psi$ can be replaced by $x \leftrightarrow \neg x_1$ and $x_1 \leftrightarrow \neg\phi \vee \neg\psi$. If $\neg\phi$ and $\neg\psi$ are not literals, equivalent formulas in negation normal form can be substituted for them, and the previous rewriting rules can be applied inductively.
   These replacements only cause a linear growth of the definitions, so the expressive power remains the same and corresponds to that of circuits according to the previous lemma. Pure and simple definitions, on the other hand, appear to be significantly weaker, since it is generally assumed that there exist Boolean circuits which require superpolynomial encodings as propositional formulas.

Besides placing restrictions on the number of literals over def-variables, we can also consider constraints on the polarity of the def-variables.

**Definition 9** (Positive Definition)

We call a definition $D(y_1, ..., y_m) = (y_1 \leftrightarrow \alpha_1, ..., y_m \leftrightarrow \alpha_m)$ **positive** if the propositional formulas $\alpha_1, ..., \alpha_m$ are each in negation normal form and contain no occurrences of negated literals over def-variables, so the only variables that can be negated are the original variables.

An example of a positive definition is

$$D(y_1, y_2) = (y_1 \leftrightarrow \neg a, y_2 \leftrightarrow (\neg b \vee y_1))$$

with $\mathit{def}_D(y_1) = \neg a$ and $\mathit{def}_D(y_2) = \neg b \vee \neg a$. From Lemma 6, it follows that $\exists y_1 \exists y_2\ (D(y_1, y_2) \wedge y_i) \approx \mathit{def}_D(y_i)$ for $i = 1, 2$. Interestingly, this property still holds when we replace the bi-implications with simple implications $\rightarrow$. Then we obtain $D'(y_1, y_2) = (y_1 \rightarrow \neg a, y_2 \rightarrow (\neg b \vee y_1))$, and it can easily be verified that $\exists y_1 \exists y_2\ (D'(y_1, y_2) \wedge y_i) \approx \mathit{def}_D(y_i)$, $i = 1, 2$. But for arbitrary definitions without restrictions on the polarities of def-variables, it is in general not possible to substitute simple implications for bi-implications without losing that property. Consider the example $G = (y_1 \leftrightarrow a, y_2 \leftrightarrow \neg y_1)$ with $\mathit{def}_G(y_1) = a$ and $\mathit{def}_G(y_2) = \neg a$. For $G' = (y_1 \rightarrow a, y_2 \rightarrow \neg y_1)$, we obtain $\exists y_1 \exists y_2\ (G' \wedge y_1) \approx a$, but $\exists y_1 \exists y_2\ (G' \wedge y_2)$ is true regardless of the value of $a$, and thus $\exists y_1 \exists y_2\ (G' \wedge y_2) \not\approx \mathit{def}_G(y_2)$.

**Lemma 10** Let $D(y_1, ..., y_m) = (y_1 \leftrightarrow \alpha_1, ..., y_m \leftrightarrow \alpha_m)$ be a positive definition, and let $D'(y_1, ..., y_m) = (y_1 \rightarrow \alpha_1, ..., y_m \rightarrow \alpha_m)$. Then

$$\exists y_1 ... \exists y_m\ (D'(y_1, ..., y_m) \wedge y_i) \approx \mathit{def}_D(y_i)$$

for $1 \le i \le m$.

**Proof** We use induction on the number of def-variables. For definitions with one def-variable, it is obvious that $\exists y_1\ ((y_1 \rightarrow \alpha_1) \wedge y_1) \approx \alpha_1$.

For $m > 1$, let $E(y_2, ..., y_m) = (y_2 \leftrightarrow \alpha_2, ..., y_m \leftrightarrow \alpha_m)$ and $E'(y_2, ..., y_m) = (y_2 \rightarrow \alpha_2, ..., y_m \rightarrow \alpha_m)$ be a subset of the (bi-)implications in $D$ and $D'$. Then

$$\begin{aligned} \Phi &:= \exists y_1 ... \exists y_m\ (D'(y_1, ..., y_m) \wedge y_i) \\ &\approx \exists y_1\ ((y_1 \rightarrow \alpha_1) \wedge (\exists y_2 ... \exists y_m\ (E'(y_2, ..., y_m) \wedge y_i)))\ . \end{aligned}$$

If $i = 1$, this is equivalent to $\exists y_1\ ((y_1 \rightarrow \alpha_1) \wedge y_1 \wedge (\exists y_2 ... \exists y_m E'(y_2, ..., y_m)))$, which in turn is equivalent to $\alpha_1$, because $\exists y_2 ... \exists y_m E'(y_2, ..., y_m)$ is trivially true by assigning $y_2 = ... = y_m = 0$.

If $i > 1$, we can expand $y_1$ by the well-known Shannon expansion:

$$\begin{aligned} & \exists y_1\ ((y_1 \rightarrow \alpha_1) \wedge (\exists y_2 ... \exists y_m\ (E'(y_2, ..., y_m) \wedge y_i))) \\ \approx\ & (\exists y_2 ... \exists y_m\ (E'(y_2, ..., y_m) \wedge y_i))\, [y_1/0] \\ & \vee (\alpha_1 \wedge (\exists y_2 ... \exists y_m\ (E'(y_2, ..., y_m) \wedge y_i)))\, [y_1/1] \end{aligned}$$

$E$ is a definition with $m-1$ def-variables, and by the induction hypothesis, the above is equivalent to

$$\begin{aligned} & \mathit{def}_E(y_i)[y_1/0] \vee (\alpha_1 \wedge \mathit{def}_E(y_i)[y_1/1]) \\ =\ & \alpha_i[y_{i-1}/\alpha_{i-1}]...[y_2/\alpha_2][y_1/0] \vee (\alpha_1 \wedge \alpha_i[y_{i-1}/\alpha_{i-1}]...[y_2/\alpha_2][y_1/1]) && (*) \\ \approx\ & \alpha_i[y_{i-1}/\alpha_{i-1}]...[y_2/\alpha_2][y_1/\alpha_1] && (**) \\ =\ & \mathit{def}_D(y_i) \end{aligned}$$

where the equivalence between $(*)$ and $(**)$ is due to the fact that $\alpha_i$ cannot contain negative occurrences of $y_1$. $\square$

It can be shown that the previously mentioned Tseitin CNF transformation ([16, 19]) by successively replacing $\alpha \vee (\beta \wedge \pi)$ with $(\alpha \vee x) \wedge (x \rightarrow (\beta \wedge \pi)) \approx (\alpha \vee x) \wedge (\neg x \vee \beta) \wedge (\neg x \vee \pi)$ for a new variable $x$ produces positive definitions. This explains why it is sufficient to use simple implications $x \rightarrow (\beta \wedge \pi)$.

In fact, it is always straightforward to obtain positive definitions: every unrestricted definition $D(y_1, ..., y_m) = (y_1 \leftrightarrow \alpha_1, ..., y_m \leftrightarrow \alpha_m)$ can be transformed into a positive definition $P(y_1^+, y_1^- ..., y_m^+, y_m^-)$ which is equivalent in the sense that $def_D(y_i) = def_P(y_i^+)$ and the length of $P$ is linear in the length of $D$. The idea is that positive occurrences of a def-variable $y_i$ are replaced with $y_i^+$, whereas $y_i^-$ represents the negated occurrences of $y_i$. Without loss of generality, we assume that $\alpha_1, ..., \alpha_m$ is in negation normal form, and with $\overline{\alpha_i}$, we denote the negation normal form of the complement $\neg\alpha_i$ of $\alpha_i$. The first step of the transformation is to replace $y_1 \leftrightarrow \alpha_1$ with $y_1^+ \leftrightarrow \alpha_1$ and $y_1^- \leftrightarrow \overline{\alpha_1}$. For $i = 2, ..., m$, we now successively define:

$$\begin{array}{lcl} y_i^+ & \leftrightarrow & \alpha_i[\neg y_{i-1}/y_{i-1}^-]...[\neg y_1/y_1^-][+y_{i-1}/y_{i-1}^+]...[+y_1/y_1^+] \\ y_i^- & \leftrightarrow & \overline{\alpha_i}[\neg y_{i-1}/y_{i-1}^-]...[\neg y_1/y_1^-][+y_{i-1}/y_{i-1}^+]...[+y_1/y_1^+] \end{array}$$

Here, $+y_k/y_k^+$ means that only unnegated occurrences of $y_k$ are replaced with $y_k^+$. Since this transformation only doubles the size of definitions, it follows that positive definitions have a similar expressiveness as unrestricted definitions.

## 6. Propositional Definitions and Quantified Boolean Formulas

The expressive power of propositional definitions is closely related to a special class of quantified Boolean formulas. Let $\Phi = \exists v_1 ... \exists v_n\ \phi_1 \wedge ... \wedge \phi_q$ be an existentially quantified Boolean formula in conjunctive normal form (∃CNF*). As before, we separate each clause $\phi_i$ into $\phi_i = \phi_i^b \vee \phi_i^f$ with the bound part $\phi_i^b$ and the free part $\phi_i^f$. Now, we place additional restrictions on the bound parts, but not on the free parts. For example, we denote with ∃HORN$^b$ the class of existentially quantified CNF formulas in which the quantified variables satisfy the Horn property. That means $\phi_i^b$ ∈HORN and $\phi_i^f$ ∈PROP, so that each clause may contain at most one positive existential literal, but arbitrarily many free or negative existential ones.

It turns out that these ∃HORN$^b$ formulas have essentially the same expressive power as definitions written as $\exists y_1 ... \exists y_m\ (y_1 \rightarrow \alpha_1) \wedge ... \wedge (y_m \rightarrow \alpha_m) \wedge y_i$ (as in Lemma 10). The latter is not necessarily a formula in conjunctive normal form, but with a combination of the distributive laws and the introduction of additional (positive) definitions as in the Tseitin procedure, it can be transformed into an equivalent CNF formula of polynomial size. Then each clause in the resulting formula has at most one negative existential literal. By inverting the polarity of each existential literal, we obtain an $\exists$HORN$^b$ formula.

An alternative transformation of propositional definitions into polynomial-size $\exists$HORN$^b$ formulas, and also in the inverse direction, is possible by the polynomial-space, and in fact also polynomial-time, transformations from definitions to Boolean circuits, and the other way round, which have been given in Lemma 8. These can be combined with results on Boolean circuits and $\exists$HORN$^b$ formulas ([1, 14]) which show that there are polynomial-time transformations between the two representations in both directions. From circuits to $\exists$HORN$^b$, the idea is to label the edges with new existentially quantified variables and to encode the meaning of the gates into clauses over the edge labels (similar to the technique in [3]). This produces $\exists$HORN$^b$ clauses which each contain at most three literals over existentially quantified variables. Circuits with fan-out 1, that means propositional formulas, can also be represented as $\exists$HORN$^b$ formulas with at most two existential literals per clause (called $\exists$2-HORN$^b$) ([5]), but it is not clear if $\exists$2-HORN$^b$ is also sufficient for circuits with arbitrary fan-out.

The transformation in the other direction from $\exists$HORN$^b$ to circuits can be performed with a generalization of unit resolution to non-clausal formulas. ([14]) The existentially quantified variables are successively eliminated by performing all possible unit resolutions on them, while simultaneously building circuits for the resolvents. Possibly exponential growth from substituting previously generated resolvents into new ones is avoided by reusing earlier results with fan-out, rather than embedding copies of previously built circuits.

In total, the combination of the transformations between propositional definitions, Boolean circuits and $\exists$HORN$^b$ formulas leads to the following theorem:

**Theorem 11** (Propositional Definitions and $\exists$HORN$^b$ Formulas)

1. For every propositional definition $D(y_1, ..., y_m)$, there exists an $\exists$HORN$^b$ formula $\Phi$ with $\Phi \approx def_D(y_m)$, such that the length of $\Phi$ and the time to compute $\Phi$ from $D$ are both polynomial in the length of $D$.
2. For every $\exists$HORN$^b$ formula $\Psi$, there exists a propositional definition $D(y_1, ..., y_m)$ with $def_D(y_m) \approx \Psi$, such that the length of $D$ and the time to compute $D$ from $\Psi$ are both polynomial in the length of $\Psi$.

We now extend definitions to quantified Boolean formulas. Let $\exists x_{i,1} ... \exists x_{i,n_i} \phi_i$, $1 \leq i \leq m$, be a formula in $\exists$CNF$^*$ and $D = (y_1 \leftrightarrow \exists x_{1,1} ... \exists x_{1,n_1} \phi_1, ..., y_m \leftrightarrow \exists x_{m,1} ... \exists x_{m,n_m} \phi_m)$. We assume that all quantified variables are distinct, i.e. $x_{i,j} \neq x_{k,l}$ whenever $i \neq k$ or $j \neq l$. Otherwise, they could be renamed accordingly. We do, however, allow variables to occur as quantified variables in one formula and as free variables in other formulas. With such definitions, we can also express formulas with more complex prefixes which are not purely existential. Consider the example $D = (y_1 \leftrightarrow \exists x_1 \phi, y_2 \leftrightarrow \exists x_2 \neg y_1, y_3 \leftrightarrow \neg y_2)$. Here, $y_1$ defines $\exists x_1 \phi$, $y_2$ is $\exists x_2 \forall x_1 \neg \phi$, and $y_3$ is $\forall x_2 \exists x_1 \phi$. Assuming that the expressive power of quantified Boolean formulas properly increases with more quantifier alternations, this allows us to define formulas which only have superpolynomial encodings as $\exists$CNF* formulas. When we restrict ourselves to positive definitions, it is easy to see that we remain in the prefix class of $\exists$CNF*.

An interesting open question is whether positive definitions with existential quantification are exponentially more expressive than positive propositional definitions or, equivalently, whether $\exists$CNF* formulas can be transformed into polynomial-size $\exists$HORN$^b$ formulas. To discuss this problem in more detail, we now introduce *universal formulas* for $\exists$CNF* formulas with at most 3 literals per clause ($\exists$3-CNF*). Let $\delta_1, ..., \delta_{t(n)}$ be the sequence of all 3-clauses over the variables $x_1, ..., x_n$ in a fixed but arbitrary order. Then the number of clauses $t(n)$ is less than $8n^3$, and for new variables $\mathbf{z} = z_1, ..., z_{t(n)}$, we define:

$$\Delta_n(z_1, ..., z_{t(n)}) := \exists x_1 ... \exists x_n \bigwedge_{1 \le i \le t(n)} (\delta_i \vee z_i)$$

Clearly, it holds for every truth assignment $v = (v(z_1), ..., v(z_{t(n)}))$ to the free variables $\mathbf{z}$ that $\Delta_n(v(z_1), ..., v(z_{t(n)})) = 1$ if and only if the set of clauses $\{\delta_i \mid v(z_i) = 0\}$ is satisfiable. Thus, the formula encodes the satisfiability problem for all formulas in 3-CNF over $n$ variables, so we call $\Delta_n(\mathbf{z})$ a universal formula for $\exists$3-CNF*.

We can now show that a polynomial-time transformation from $\exists$CNF* to its subclass $\exists$HORN$^b$ would imply P = NP.

**Lemma 12** If P $\neq$ NP, there exists no polynomial $p$ such that for every $\exists$CNF* formula $\Phi$, an equivalent $\exists$HORN$^b$ formula $\Psi$ can be computed in time at most $p(|\Phi|)$.

**Proof** We assume the contrary that a polynomial-time transformation from $\exists$CNF* to $\exists$HORN$^b$ exists and use it to construct a polynomial-time algorithm for the NP-complete satisfiability problem for 3-CNF.

Let $\alpha = \alpha_1 \wedge ... \wedge \alpha_k$ be an arbitrary 3-CNF formula over the variables $x_1, ..., x_n$, and let $\Delta_n(z_1, ..., z_{t(n)}) = \exists x_1 ... \exists x_n \bigwedge_{1 \le i \le t(n)} (\delta_i \vee z_i)$ be the corresponding universal formula for $n$ variables. For sake of simplicity, we assume $\alpha_i = \delta_i$ for $1 \le i \le k$, and we let $v = (v(z_1), ..., v(z_{t(n)}))$ be a truth assignment to the free variables with $v(z_1) = ... = v(z_k) = 0$ and $v(z_{k+1}) = ... = v(z_{t(n)}) = 1$.

Now, $\alpha$ is satisfiable if and only if $\exists x_1 ... \exists x_n \ (\delta_1 \wedge ... \wedge \delta_k) = 1$, which in turn holds if and only if $\Delta_n(v(z_1), ..., v(z_{t(n)})) = 1$. According to our assumption, we can compute in polynomial time an $\exists$HORN$^b$ formula $\Psi_n$ with $\Delta_n \approx \Psi_n$, which means $\Delta_n(v(z_1), ..., v(z_{t(n)})) = 1$ if and only if $\Psi_n(v(z_1), ..., v(z_{t(n)})) = 1$. But in the formula $\Psi(v(z_1), ..., v(z_{t(n)}))$, all free variables are replaced with their corresponding truth values, which means only quantified variables are left, and they must satisfy the Horn property. An existentially quantified Horn formula can be solved in linear time with the well-known unit resolution for propositional Horn formulas, and thus $\mathrm{P} = \mathrm{NP}$ in contradiction to our assumption. □

Since it is widely believed that P does not equal NP, we can assume that there is no polynomial-time transformation from $\exists$CNF$^*$ formulas to equivalent $\exists$HORN$^b$ formulas. The more general question is whether all $\exists$CNF$^*$ formulas have equivalent polynomial-size $\exists$HORN$^b$ formulas when we allow transformations with unlimited time constraints. It is strongly conjectured that the classes have different expressive power. In that case, the universal formulas $\Delta_n$ separate both classes.

**Lemma 13** There exist $\exists$CNF$^*$ formulas without a polynomial-size $\exists$HORN$^b$ equivalent if and only if there is no polynomial $p$ such that for every $n$ there is a formula $\Psi_n \in \exists$HORN$^b$ with $\Psi_n \approx \Delta_n$ and $|\Psi_n| \leq p(|\Delta_n|)$.

**Proof** The direction from right to left is clear. For the other direction, we show that if such a polynomial exists, we can transform an arbitrary $\exists$CNF$^*$ formula $\Phi(\mathbf{w}) = \exists x_1 ... \exists x_n \bigwedge_{1 \leq i \leq q} \phi_i(\mathbf{x}, \mathbf{w})$ into a polynomial-size $\exists$HORN$^b$ formula. As before, we consider each clause $\phi_i(\mathbf{x}, \mathbf{w})$ to consist of a bound part $\phi_i^b(\mathbf{x})$ and a free part $\phi_i^f(\mathbf{w})$. We assume that both parts are non-empty for every clause, because clauses without bound variables could be moved in front of the prefix and later be added to the $\exists$HORN$^b$ formula that we are going to construct. Clauses $\phi_j = \phi_j^b$ without free part can be replaced with $(\phi_j^b \vee w_1) \wedge (\phi_j^b \vee \neg w_1)$. If two clauses have the same bound parts, i.e. $(\phi_i^b \vee \phi_i^f) \wedge (\phi_i^b \vee \phi_j^f)$ for $i \neq j$, we can replace the first clause with $(y \vee \phi_i^f) \wedge (\neg y \vee \phi_i^b \vee \phi_i^f)$ for a new existentially quantified variable $y$, so it is sufficient to consider only formulas with pairwise disjoint bound parts. We can also assume that all bound parts are in 3-CNF. Otherwise, we could introduce new existential variables to split bound parts, e.g. $(l_1 \vee l_2 \vee l_3 \vee l_4 \vee \phi_i^f) \approx \exists y\ (l_1 \vee l_2 \vee y \vee \phi_i^f) \wedge (\neg y \vee l_3 \vee l_4 \vee \phi_i^f)$.

For each clause $(\phi_i^b \vee \phi_i^f)$ in $\Phi(\mathbf{w})$, there is a corresponding clause $(\delta_{i'} \vee z_{i'})$ with $\delta_{i'} = \phi_i^b$ in $\Delta_n(\mathbf{z})$. We now replace $(\phi_i^b \vee \phi_i^f)$ with $(\phi_i^b \vee z_{i'})$ for all $i = 1...q$ and obtain (after eliminating duplicate clauses) a formula $\Phi'(\mathbf{z})$ which is a subformula of $\Delta_n(\mathbf{z})$. According to our assumption, there is a $\exists$HORN$^b$ formula $\Psi_n(\mathbf{z})$ equivalent to $\Delta_n(\mathbf{z})$ of length at most $p(|\Delta_n|)$, and thus also polynomial in the length of $\Phi$. If all free variables $z_j$ which do not occur in $\Phi'$ are assigned in $\Psi_n$ the value 1, the simplified formula $\Psi_n^*$ is equivalent to $\Phi'$.

Now, we undo the previous substitution of free variables $z_{i'}$ for free parts $\phi_i^f(\mathbf{w})$. The resulting $\exists$HORN$^b$ formula is equivalent to $\Phi$ and polynomial in the length of $\Phi$, so we have a contradiction to the claim that we cannot always obtain short equivalent $\exists$HORN$^b$ formulas. □

**Definition 14** A formula class $A$ is said to be **poly-size closed under negation** if and only if there is a polynomial $p$, such that for every formula $\Phi \in A$ there exists a formula $\Psi \in A$ with $\Psi \approx \neg\Phi$ and $|\Psi| \leq p(|\Phi|)$.

The class $\exists$HORN$^b$ is poly-size closed under negation. That can be seen as follows: First, we transform a given formula $\Phi \in \exists$HORN$^b$ into an equivalent polynomial-size circuit $c$ using the previously mentioned transformation from [1, 14]. Then we negate the circuit and transform $\neg c$ into a circuit $c'$ in standard form where only inputs are negated. This can be achieved by applying de Morgan's laws and the elimination of multiple subsequent negations. A circuit in standard form can then again be represented as a polynomial-size $\exists$HORN$^b$ formula. The transformations require at most polynomial time and length, so we obtain our desired result. For $\exists$CNF$^*$, it is again an open question whether the class is also poly-size closed under negation. The next lemma shows that we cannot expect to find such complements within $\exists$CNF$^*$ in polynomial time.

**Lemma 15** Assuming NP $\neq$ coNP, there is no polynomial-time algorithm to compute for every given formula $\Phi \in \exists$CNF$^*$ a formula $\Psi \in \exists$CNF$^*$ equivalent to $\neg\Phi$.

**Proof** Let $\phi = \phi_1 \wedge ... \wedge \phi_q$ be an arbitrary propositional CNF formula over variables $x_1, ..., x_n$. Then $\phi$ is unsatisfiable if and only if $\Phi := \exists x_1 ... \exists x_n\ \phi_1 \wedge ... \wedge \phi_q$ is false, which in turn holds if and only if $\neg\Phi$ is true. If we could compute in polynomial time an $\exists$CNF$^*$ formula $\Psi = \exists y_1 ... \exists y_m\ \psi_1 \wedge ... \wedge \psi_s$ with $\Psi \approx \neg\Phi$, we could solve the coNP-complete unsatisfiability problem for propositional formulas in nondeterministic polynomial time by searching for a satisfying assignment in the complement $\Psi$, which would imply NP=coNP. $\square$

Similar to Lemma 13 in which the universal formulas $\Delta_n$ separate the classes $\exists$HORN$^b$ and $\exists$CNF$^*$, if they are distinct, the negations of universal formulas have a superpolynomial length in $\exists$CNF$^*$ if the class is not poly-size closed under negation. The proof is analogous to the proof of Lemma 13.

**Lemma 16** The class $\exists$CNF$^*$ is not poly-size closed under negation if and only if there are universal formulas $\Delta_n$ for which the shortest representation of $\neg\Delta_n$ in $\exists$CNF$^*$ requires superpolynomial length.

## 7. Nested Boolean Functions: Definitions as Function Schemes

An alternative approach for introducing definitions is based on a more functional view. Every propositional or quantified Boolean formula can be interpreted as a Boolean function. For example, $(\neg x \vee y) \wedge (x \vee \neg y \vee z)$ describes a function $f(x, y, z)$ over variables $x$, $y$ and $z$. A further abstraction is to treat $f(x, y, z) := (\neg x \vee y) \wedge$

$(x \vee \neg y \vee z)$ as a scheme of formulas having the same structure. Then $f(x, y, z)$ can be instantiated with different variables or other functions. For example, we denote with $f(a, b, c)$ the formula $(\neg a \vee b) \wedge (a \vee \neg b \vee c)$, and $f(g(u, v), v, h(w))$ is the formula $(\neg g(u, v) \vee v) \wedge (g(u, v) \vee \neg v \vee h(w))$. We also allow instantiations with truth values, such as $f(1, b, c) = (\neg 1 \vee b) \wedge (1 \vee \neg b \vee c)$.

In [7], such a sequence of Boolean functions is called a Boolean program. The term is also used for a different concept in the context of verification, so to avoid confusion, we prefer to speak of *nested Boolean functions* (NBF). An interesting example given in [7] is the computation of parity. Let

$$f_0(p_1, p_2) := (\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2)$$

be the parity of two binary variables. Then the parity of four variables can be computed by reusing $f_0$:

$$f_1(p_1, p_2, p_3, p_4) := f_0(f_0(p_1, p_2), f_0(p_3, p_4))$$

Now, the parity of 16 variables is

$$f_2(p_1, ..., p_{16}) := f_1(f_1(p_1, ..., p_4), f_1(p_5, ..., p_8), f_1(p_9, ..., p_{12}), f_1(p_{13}, ..., p_{16}))$$

and so on. Obviously, computing the parity of $m = 2^{2^n}$ variables requires only $n + 1$ function definitions, and the longest contains $O(m)$ variable or function symbols. It is well known [12, 20] that an equivalent propositional formula would have length at least $m^2$. Before we further discuss the expressive power of nested Boolean functions, we give a formal inductive definition.

**Definition 17** (Nested Boolean Function)

A nested Boolean function NBF (or Boolean program [7]) is a finite sequence of Boolean functions $F = (f_1, ..., f_l)$. Each $f_i$ has an associated arity $n_i$ and arguments $\mathbf{x}_i := x_{i,1}, ..., x_{i,n_i}$.

For some $t \in \{1, ..., l\}$, the initial functions $f_1, ..., f_t$ are each defined by a propositional formula $\alpha_i(\mathbf{x}_i)$, that is, $f_i(x_{i,1}, ..., x_{i,n_i}) := \alpha_i(x_{i,1}, ..., x_{i,n_i})$ for $i = 1, ..., t$.

For $i > t$, $f_i$ can be composed of previously defined functions $f_{j_0}, ..., f_{j_m}$ with $j_0, ..., j_m \in \{1, ..., i-1\}$. Let $f_{j_0}$ have arity $m$, then the definition of $f_i$ has the form $f_i(\mathbf{x}_i) := f_{j_0}(f_{j_1}(\mathbf{y}_1), ..., f_{j_m}(\mathbf{y}_m))$, where $\mathbf{y}_1, ..., \mathbf{y}_m$ are tuples of matching arities over variables in $\mathbf{x}_i$ or the Boolean constants 0 and 1.

The length $|F|$ of a nested Boolean function $F = (f_1, ..., f_l)$ is the total number of occurrences of variable or function symbols in the definitions of $f_1, ..., f_l$.

Nested Boolean functions can provide a characterization of PSPACE by encoding computations of polynomial space Turing machines into NBFs of polynomial size, and vice versa [7]. Alternatively, it is possible to give a transformation from quantified Boolean formulas to NBF by simulating quantifiers. The idea is as follows: let $\exists x\ \phi(x, z_1, ..., z_r)$ be an existentially quantified Boolean formula with a propositional matrix $\phi$ over the existential $x$ and free variables $z_1, ..., z_r$. Then $\exists x\ \phi(x, z_1, ..., z_r) \approx \phi(0, z_1, ..., z_r) \vee \phi(1, z_1, ..., z_r)$ by the well-known Shannon expansion. This can easily be expressed by a nested Boolean function: if we define $f(x, z_1, ..., z_r) := \phi(x, z_1, ..., z_r)$ and $g_\exists(a, b) := a \vee b$, it follows that $g_\exists(f(0, z_1, ..., z_r), f(1, z_1, ..., z_r))$ is equivalent to the existentially quantified formula. In propositional logic, we would have to explicitly write down $\phi(0, z_1, ..., z_r)$ and $\phi(1, z_1, ..., z_r)$ as propositional formulas, which would cause exponential growth when applied multiple times, but nested Boolean functions avoid this through definitions and instantiations. Universal quantifiers can be handled analogously be the dual expansion $\forall x\ \phi(x, z_1, ..., z_r) \approx \phi(0, z_1, ..., z_r) \wedge \phi(1, z_1, ..., z_r)$.In total, we obtain the following transformation: let $\Phi(\mathbf{z}) = Q_n v_n ... Q_1 v_1\ \phi(v_1, ..., v_n, \mathbf{z})$, $Q_i \in \{\forall, \exists\}$, be a QBF formula with quantified variables $v_1, .., v_n$ and free variables $\mathbf{z} = z_1, ..., z_r$. Then we define

$$\begin{aligned}
&f_0(v_1, ..., v_n, \mathbf{z}) := \phi(v_1, ..., v_n, \mathbf{z})\\
&g_\forall(a, b) := a \wedge b\\
&g_\exists(a, b) := a \vee b
\end{aligned}$$

and for $i = 1, ..., n$, we let

$$f_i(v_{i+1}, ..., v_n, \mathbf{z}) := g_{Q_i}(f_{i-1}(0, v_{i+1}, ..., v_n, \mathbf{z}), f_{i-1}(1, v_{i+1}, ..., v_n, \mathbf{z}))$$

where $g_{Q_i}$ is $g_\forall$ if $Q_i = \forall$ and $g_\exists$ otherwise.

Clearly, the function $f_n(\mathbf{z})$ is equivalent to $\Phi(\mathbf{z})$, and the length of the NBF $F = (f_0, g_\exists, g_\forall, f_1, ..., f_n)$ is quadratic in the length of $\Phi(\mathbf{z})$, so we have the following lemma:

**Lemma 18** For every quantified Boolean formula $\Phi(\mathbf{z})$, there exists a nested Boolean function $F = (f_1, ..., f_l)$, such that $f_l(\mathbf{z}) \approx \Phi(\mathbf{z})$ and $|F| = O(|\Phi(\mathbf{z})|^2)$.

A transformation in the other direction from NBF to QBF* is also possible due to the fact that for every nested Boolean function $F = (f_1, ..., f_l)$ over variables $\mathbf{z}$, there exists a polynomial-size Turing machine $M$ with input $\mathbf{z}$ which accepts the input if and only if $f_l(\mathbf{z}) = 1$ and which can in turn be mapped in polynomial time to a QBF* formula with free variables $\mathbf{z}$ [7, 15].

## 8. Conclusion

The ability to have auxiliary variables which are not directly taken into account when considering logical equivalence is a powerful tool. This allows introducing definitions to avoid repetitions and to increase clarity. We have seen that propositional definitions correspond to a class of existentially quantified Boolean formulas where the bound variables satisfy the Horn property, namely $\exists$HORN$^b$, and Boolean circuits with arbitrary fan-out. All of these representations are exponentially more expressive than propositional CNF. It is widely assumed, but still not proven, that arbitrary propositional formulas, or circuits with fan-out 1, are also exponentially less powerful than circuits with arbitrary fan-out and the other representations with auxiliary variables. A similar open question that we have considered is whether $\exists$CNF$^*$ is exponentially more expressive than $\exists$HORN$^b$.

Another very powerful feature appears to be the addition of universal quantifiers, which we have seen to be equivalent to nested Boolean functions or positive and negative definitions with existential quantifiers. It is currently not well understood how universal quantification can be used in practice to obtain concise encodings, and we hope that considering alternate representations like nested Boolean functions also leads to interesting new encoding patterns for QBF$^*$.

## References

[1] S. Aanderaa and E. Börger, 1979, "The Horn complexity of Boolean functions and cook's problem", *Proc. 5th Scandinavian Logic Symposium 1979*, Aalborg University Press, pp. 231-256.

[2] B. Aspvall, M. Plass and R. Tarjan, 1979, "A linear-time algorithm for testing the truth of certain quantified Boolean formulas", *Information Processing Letters*, 8(3):121-123.

[3] M. Bauer, D. Brand, M. Fischer, A. Meyer and M. Paterson, 1973, "A note on disjunctive form tautologies", *SIGACT News*, 5(2):17-20.

[4] A. Biere, A. Cimatti, E. Clarke and Y. Zhu, 1999, "Symbolic model checking without BDDs", *Proc. 5th Intl. Conf. on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1999)*, Springer LNCS 1579, pp. 193-207.

[5] U. Bubeck, and H. Kleine Büning, 2009, "A new 3-CNF transformation by parallel-serial graphs", *Journal Information Processing Letters*, 109(7):376-379.

[6] S. Cook, 1976, "A short proof of the pigeon hole principle using extended resolution", *SIGACT News*, 8(4):28-32.

[7] S. Cook and M. Soltys, 1999, "Boolean programs and quantified propositional proof systems", *The Bulletin of the Section of Logic*, 28(3):119-129.

[8] A. Flögel, M. Karpinski and H. Kleine Büning, 1995, "Resolution for quantified Boolean formulas", *Information and Computation*, 117(1):12-18.

[9] A. Flögel, H. Kleine Büning, and T. Lettmann, 1993, "On the restricted equivalence for subclasses of propositional logic", *RAIRO Theoretical Informatics and Applications*, 27(4):327-340.

[10] A. Haken, 1985, "The intractability of resolution",*Theoretical Computer Science*, 39(2-3):297-308.

[11] H. Kautz and B. Selman, 1992, "Planning as satisfiability",*Proc. 10th European Conf. on Artificial Intelligence (ECAI 1992)*, pp. 359-363.

[12] V. Khrapchenko, 1972, "Methods of determining lower bounds for the complexity of $\pi$-schemes", *Mat. Zametki*, 10(1):83-92 (in Russian); English translation in: *Math. Notes Acad. Sciences USSR*, 10(1):474-479.

[13] H. Kleine Büning and T. Lettmann, 1999, *Propositional Logic: Deduction and Algorithms*, Cambridge University Press, Cambridge, UK.

[14] H. Kleine Büning, X. Zhao and U. Bubeck, 2009, "Resolution and expressiveness of subclasses of quantified Boolean formulas and circuits", *Proc. 12th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2009)*, Springer LNCS 5584, pp. 391-397.

[15] A. Meyer and L. Stockmeyer, 1973, "Word problems requiring exponential time", *Preliminary Report, Proc. 5th ACM Symp. on Theory of Computing (STOC 1973)*, pp. 1-9.

[16] D. Plaisted and S. Greenbaum, 1986, "A structure-preserving clause form translation", *Journal of Symbolic Computation*, 2(3):293-304.

[17] J. Robinson, 1965, "A machine-oriented logic based on the resolution principle", *Journal of the ACM*, 12(1):23-41.

[18] L. Stockmeyer, 1976, "The polynomial-time hierarchy", *Theoretical Computer Science*, 3(1):1-22.

[19] G. Tseitin, 1970, "On the Complexity of Derivation in Propositional Calculus", In A. Silenko (ed.): *Studies in Constructive Mathematics and Mathematical Logic*, Part II, pp. 115-125.

[20] I. Wegener, 1987, *The Complexity of Boolean Functions*, Wiley-Teubner Series in Computer Science, B.G. Teubner, Stuttgart.

[21] C. Wrathall, 1976, "Complete sets and the polynomial-time hierarchy", *Theoretical Computer Science*, 3(1):23-33.

# 辅助变元之于命题与量化布尔公式的作用

乌维布贝克
帕德博恩大学计算机科学研究所
bubeck@upb.de
汉斯克莱那布吕宁
帕德博恩大学计算机科学研究所
kbcsl@upb.de

## 摘 要

为了精简且自然流畅的表达知识，现今学界多用借辅助变元以引入定义的方法。这篇论文中，我们给出了辅助变元的形式化定义，检验了其表达力并讨论了有趣的相关应用。

我们把以下两者联系起来：一是，反复使用未复制定义的中间结果的想法；二是，布尔函数其他表现中的相似概念。特别的，我们证明了包含相关定义的命题逻辑与以下两者具有相同的表达力。一是具有任意输出端的布尔线路，二是约束变元满足Horn性质的存在量化布尔公式（记作$\exists$HORN$^b$）。

本文还考虑了定义结构的限制，以及命题定义的扩充。特别的，我们检验了正命题定义与存在量化的正定义之间的关系（或等价地，检验了$\exists$HORN$^b$公式和约束变元未被Horn限定的存在量化的CNF公式（记作$\exists$CNF$^*$）之间的关系）。对命题定义的进一步扩充，是允许在定义中使用任意的量词，或等价地，允许布尔公式的嵌入。我们还证明了，量化CNF公式其约束变元的表达力，是由闭包中被约束部分的极小不满足子公式或极小假子公式的结构所决定的。