# Transformations into Normal Forms for Quantified Circuits *

Hans Kleine Büning[1], Xishun Zhao[2], and Uwe Bubeck[1]

[1] Computer Science Institute, University of Paderborn, Germany
kbcsl@upb.de (H. Kleine Büning), bubeck@upb.de (U. Bubeck)
[2] Institute of Logic and Cognition, Sun Yat-sen University, PR China
hsszxs@mail.sysu.edu.cn (X. Zhao)

**Abstract.** We consider the extension of Boolean circuits to quantified Boolean circuits by adding universal and existential quantifier nodes with semantics adopted from quantified Boolean formulas (QBF). The concept allows not only prenex representations of the form $\forall x_1 \exists y_1 ... \forall x_n \exists y_n\, c$ where $c$ is an ordinary Boolean circuit with inputs $x_1, ..., x_n, y_1, ..., y_n$. We also consider more general non-prenex normal forms with quantifiers inside the circuit as in non-prenex QBF, including circuits in which an input variable may occur both free and bound. We discuss the expressive power of these classes of circuits and establish polynomial-time equivalence-preserving transformations between many of them. Additional polynomial-time transformations show that various classes of quantified circuits have the same expressive power as quantified Boolean formulas and Boolean functions represented as finite sequences of nested definitions (NBF). In particular, universal quantification can be simulated efficiently by circuits containing only existential quantifiers if overlapping scopes of variables are allowed.

## 1 Introduction

Boolean circuits are a powerful concept to store propositional formulas. On the one hand, they can suitably represent important structural information, and on the other hand, they allow sharing of common subexpressions. For example, a formula like $(\alpha_1 \vee \beta_1) \wedge (\alpha_1 \vee \beta_2) \wedge (\alpha_2 \vee \beta_1) \wedge (\alpha_2 \vee \beta_2)$ with arbitrary subformulas $\alpha_1, \alpha_2, \beta_1, \beta_2$ can be represented by a circuit in which the value of each of those subformulas is computed only once and then reused multiple times by fanout. Fig. 1 shows a simple circuit for the formula $(\neg x \vee (x \wedge \neg y)) \wedge ((x \wedge \neg y) \vee z)$ in which $(x \wedge \neg y)$ is shared. Under favorable circumstances, sharing can lead to significantly shorter encodings of formulas as circuits. Accordingly, circuits have been used successfully also for SAT and QBF solvers, e.g. in [9, 10, 11, 13].

Similar to other representations like [14], Boolean circuits can be extended by allowing quantifiers. That is, a circuit can contain universal and existential quantifier nodes in addition to propositional logic gates. Typically, existing work on

* Research partially supported by DFG grant KL 529/QBF and NSFC grant 60970040.

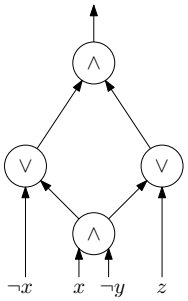


Fig. 1: Circuit in negation normal form

quantified circuits, e.g. [2], has focused on decision problems for circuits in prenex form $\forall x_1 \exists y_1 ... \forall x_n \exists y_n \ c(x_1, y_1, ..., x_n, y_n)$, where $c$ is an ordinary quantifier-free Boolean circuit and quantification is applied subsequently on its input variables. In this paper, we want to investigate the expressive power of quantified circuits not only in prenex form, but also in arbitrary negation normal form where quantifiers are allowed inside the circuit. The most general case that we consider covers circuits in which an input variable may occur both free and bound. In QBF, it is easy to obtain unique variable names and prenex representations by renaming bound variables. We will see that such renamings are problematic for quantified circuits due to the sharing of subcircuits, but this paper shows that there still exist efficient transformations between many classes of quantified circuits. Another main contribution is the interesting result that universal quantification can be simulated in linear time by negation normal form circuits containing only existential quantifiers if overlapping variable scopes are allowed. For QBF, this appears to be possible only for special classes such as quantified Horn or 2-CNF formulas [3, 5], unless the polynomial hierarchy collapses.

## 2 Boolean Circuits and Propositional Formulas

We begin our discussion with a brief review of basic relationships between circuits and propositional formulas. A *Boolean circuit* is a directed acyclic graph with one outgoing edge (the *sink*) and multiple input nodes labeled with Boolean variables. The other nodes are AND-, OR-, and NOT-gates that each have two (AND and OR) or one (NOT) incoming edges and an arbitrary number of outgoing edges. We let $\mathcal{C}$ be the class of circuits in *negation normal form* (NNF), that means circuits in which the inner nodes are only AND- and OR-gates and the inputs are variables $x$ and negated variables $\neg x$. Fig. 1 shows an example of a circuit in NNF. The *length* or *size of a circuit* is the number of gates, including negations associated with input variables. By the laws of De Morgan and the elimination of double negations, any circuit can be transformed in linear time in NNF, although in the worst case the size of the circuit might double. Subsequently, we consider only circuits in NNF, because we later want to avoid quantifier nodes being negated. The negation of a quantifier essentially inverts

its meaning, which is in particular problematic in combination with subcircuit sharing, because one quantifier might then be shared in its original and in its negated form, so a variable would be existentially and universally quantified in the same subcircuit. For clarity, we sometimes use the equivalence operator (written as $=$). But since operations such as equivalence, and also implication or XOR, implicitly contain negation and therefore cause the same problems with quantifiers, we consider such operations only as abbreviations which must be expanded in the actual representation of the circuit.

For propositional formulas, we allow the same operators $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT) and propositional variables. Similar to circuits, a formula is in negation normal form if the negation symbols occur only directly in front of variables. The *length of a formula* is the number of variable occurrences.

**Definition 1.** *(Propositional Formulas and Circuits)*
*1.* BF $:= \{\phi \mid \phi$ *is propositional formula in negation normal form*$\}$
*2.* $\mathcal{C} := \{c \mid c$ *is a Boolean circuit in negation normal form*$\}$

Let $\alpha$ and $\beta$ be two formulas or two circuits or a mixed pair of one formula and one circuit, and let $z_1, ..., z_r$ be the union of all variables which occur in $\alpha$ or $\beta$ inside a formula or in a circuit input. Then $\alpha$ and $\beta$ are *(logically) equivalent*, in symbols $\alpha \approx \beta$, if and only if for every truth assignment to $z_1, ..., z_r$ it holds that $\alpha$ and $\beta$ evaluate to the same truth value after substituting the assigned truth values for $z_1, ..., z_r$. This is significantly stronger than *satisfiability equivalence*, which requires that if one of the formulas or circuits is satisfied by some assignment to $z_1, ..., z_r$, the other one must also have some (possibly different) satisfying assignment to $z_1, ..., z_r$. Satisfiability equivalence is sometimes too weak: for example, replacing a term inside a larger formula with a different term is in general only sound if both terms are logically equivalent.

For circuit or formula classes $A$ and $B$, we write $A \leq_p B$ to express that there are polynomial-time transformations from $A$ to $B$. That means $A \leq_p B$ if and only if there is a poly-time mapping $T$, such that $T(a) \in B$ and $T(a) \approx a$ for each $a \in A$. $A =_p B$ is an abbreviation for $A \leq_p B$ and $B \leq_p A$.

There is obviously a close relationship between Boolean circuits and propositional formulas: Every propositional formula can be considered as a circuit with fanout 1, and vice versa. Fanout 1 means that every node has exactly one outgoing edge and there is no sharing of subcircuits. On the other hand, an arbitrary Boolean circuit can be encoded as a formula when we label the edges of the circuit with new auxiliary variables and describe the gates by propositional clauses over these auxiliary variables [1]. For example, we obtain $y = x_1 \wedge x_2$ for an AND-node having incoming edges labeled with $x_1$ and $x_2$ and output edges labeled with $y$. This can be performed in linear time, but the resulting formula is in general only satisfiability equivalent to the circuit, because adding new variables typically destroys logical equivalence. To achieve full logical equivalence, the auxiliary variables must be bound by existential quantifiers, which are formally introduced in the following section.

## 3 Quantified Boolean Formulas

*Quantified Boolean formulas* (QBF) extend propositional logic with universal ($\forall$) and existential ($\exists$) quantifiers over variables. For example, $\forall x(\neg x \vee (\exists y(y \vee x \vee z)))$ is a quantified Boolean formula. Variables on which a quantifier is applied are called *bound* variables, and variables which are not bound by a quantifier are *free*. In the example, $x$ and $y$ are bound, and $z$ is free. $\forall x\ \phi(x)$ is defined to be true if and only if $\phi(0)$ is true *and* $\phi(1)$ is true, and $\exists y\ \phi(y)$ means that $\phi(0)$ *or* $\phi(1)$ is true. To save parentheses, we assume that the logical connectives have a higher binding priority than the quantifiers, so we could also write the previous example as $\forall x\ \neg x \vee (\exists y\ y \vee x \vee z)$. As for propositional formulas, the length of a quantified Boolean formula is the number of variable occurrences, but now including occurrences with quantifiers. Accordingly, the example has length 6.

A formula $\Phi(z_1, ..., z_r)$ with free variables $z_1, ..., z_r$ is *satisfiable* if and only if there exists a truth assignment $\tau$ to the free variables such that $\Phi(\tau(z_1), ..., \tau(z_r))$ is true. Here, $\Phi(\tau(z_1), ..., \tau(z_r))$ denotes the substitution of the truth values in $\tau$ for the free variables in $\Phi$. Two quantified Boolean formulas $\Phi(z_1, ..., z_r)$ and $\Psi(w_1, ..., w_s)$ with free variables $z_1, ..., z_r$ and $w_1, ..., w_s$ are logically equivalent if and only if for every truth assignment $\tau$ to the free variables $z_1, ..., z_r, w_1, ..., w_s$ both formulas evaluate to the same truth value. This means that the bound variables are not directly considered when checking for logical equivalence, which makes them local to the respective formula. The ability to introduce new local variables without losing full equivalence is a powerful advantage over ordinary propositional calculus. A propositional formula can be considered as a special case of a quantified Boolean formula in which all variables are free. Similarly, all input variables in a Boolean circuit can be treated as free variables. Logical equivalence between QBF formulas is then a generalization of the equivalence criterion presented in the previous section and can naturally be extended to mixed pairs from all three representations.

Negation normal form is defined for QBF as it is for propositional formulas and can be achieved with the additional equivalences $\neg(\forall x\ \Phi) \approx \exists x \neg\Phi$ and $\neg(\exists x\ \Phi) \approx \forall x \neg\Phi$. A QBF formula $\Phi$ is in *prenex form* if $\Phi = Q_1 v_1 ... Q_k v_k\ \phi$ with quantifiers $Q_i \in \{\forall, \exists\}$ and a propositional formula $\phi$. We call $Q := Q_1 v_1 ... Q_k v_k$ the *prefix* and $\phi$ the *matrix* of $\Phi$.

**Definition 2.** *(Quantified Boolean Formulas)*
*1.* QBF := $\{\Phi \mid \Phi$ *is a quantified Boolean formula in negation normal form*$\}$
*2.* $\exists$BF := $\{\Phi \mid \Phi \in$ QBF *and* $\Phi$ *contains only existential quantifiers*$\}$
*3.* pQBF := $\{\Phi \mid \Phi \in$ QBF *in prenex form*$\}$

An arbitrary QBF can be transformed in linear time into prenex form. The result is in general not unique, and different prenexing strategies have been widely investigated, e.g. in [8]. Following the notation $\leq_p$ and $=_p$, we write $\leq_{linear}$ and $=_{linear}$ for linear-time transformations.

**Proposition 1.** QBF $=_{linear}$ pQBF

## 4 Nested Boolean Functions

Before we introduce quantified circuits, we briefly consider another representation of Boolean functions which appears rather different at first glance, but will soon turn out to be closely related to both quantified circuits and QBF. Every Boolean function can be defined by a suitable set of initial functions, for example AND, OR and NOT, and a composition of these functions. Instead of a fixed set of starting functions, we can also allow arbitrary propositional formulas as starting functions. In order to illustrate the idea, we present a short example:
Let $f_1(x, y) := x \vee \neg y$ and $f_2(z, w) := z \wedge w$ be two initial functions, and let
$f_3(x_1, x_2) := f_2(0, f_1(x_1, x_2))$ and
$f_4(x, y, z) := f_2(f_3(x, z), f_1(z, y))$ be compound functions.
Then $f_4(x, y, z)$ is equivalent to the propositional formula $(0 \wedge (x \vee \neg z)) \wedge (z \vee \neg y)$.
Such definition schemes for Boolean functions have been introduced in [6] as *Boolean Programs*. But this name is also used for different concepts in other fields. To avoid confusion, we use the name *Nested Boolean Functions* (NBF).

**Definition 3.** *A* nested Boolean function (NBF) *is a finite sequence* $D(f_k) = (f_1, ..., f_k)$ *of definitions of Boolean functions. For fixed* $t \in \{1, ..., k\}$*, it contains*

- initial functions $f_1, ..., f_t$*, which are each defined by* $f_i(\mathbf{x}^i) := \alpha_i(\mathbf{x}^i)$ *for a propositional formula* $\alpha_i$ *over variables* $\mathbf{x}^i := (x^{i,1}, ..., x^{i,n_i})$*, and*
- compound functions $f_{t+1}, ..., f_k$ *of the form* $f_i(\mathbf{x}^i) := f_{j_0}(f_{j_1}(\mathbf{x}^i_1), ..., f_{j_r}(\mathbf{x}^i_r))$ *for previously defined functions* $f_{j_0}, ..., f_{j_r} \in \{f_0, ..., f_{i-1}\}$*. The arguments* $\mathbf{x}^i_1, ..., \mathbf{x}^i_r$ *are tuples containing variables in* $\mathbf{x}^i$ *or Boolean constants, such that the arity of* $\mathbf{x}^i_l$ *matches the arity of* $f_{j_l}$ *and* $r$ *is the arity of* $f_{j_0}$*.*

*We call* $f_k$ *the* defined Boolean function*. The* length of a NBF $D(f_k)$ *is* $|D(f_k)| := |f_1| + ... + |f_k|$*, where* $|f_i|$ *is the total number of occurrences of constants, variables and function symbols on the right hand side of the defining equation of* $f_i$*.*

A Boolean circuit $c(\mathbf{x})$ with input variables $\mathbf{x}$ can be represented as a finite sequence of definitions $D(f_c(\mathbf{x})) \in$ NBF with $c(\mathbf{x}) \approx f_c(\mathbf{x})$. The initial functions are $f_{id}(x) := x$, $f_\neg(x) := \neg x$, $f_\wedge(x, y) := x \wedge y$ and $f_\vee(x, y) := x \vee y$. Bottom-up, we assign a function to each edge in the circuit. For an input $x$ or $\neg x$, we simply use $f_{id}(x)$ or $f_\neg(x)$. For the output edges of an AND-node over incoming edges associated with $g(\mathbf{x})$ and $h(\mathbf{y})$, we choose $f(\mathbf{x} \cap \mathbf{y}) := f_\wedge(g(\mathbf{x}), h(\mathbf{y}))$. Here, $\mathbf{x} \cap \mathbf{y}$ is the tuple of all variables occurring in $g$ and $h$, without multiple occurrences and in arbitrary order. Analogously, we assign functions to the OR-nodes and NOT-nodes (for non-NNF circuits). It is not difficult to see that the function associated with the outgoing edge of the circuit is equivalent to the circuit. We have already mentioned a linear-time encoding of circuits as existentially quantified Boolean formulas (Section 2). Now, the number of function symbols in the resulting NBF is again linear in the size of the circuit, but the length of the NBF also includes occurrences of variables as arguments. We obtain a time and space bound of $O(|\mathbf{x}| \cdot |c(\mathbf{x})|)$, where $|\mathbf{x}|$ is the number of input variables and $|c(\mathbf{x})|$ is the circuit size. Subsequently, we use the term *v-linear* for $O(|v| \cdot |A|)$, where $|A|$ is the length of an expression $A$ and $|v|$ the number of variables in $A$.

**Proposition 2.** $\mathcal{C} \leq_{v\text{-}linear}$ NBF

For an arbitrary NBF $D(f_k) = (f_1, ..., f_k)$, the problem of deciding whether $f_k(a_1, ..., a_{n_k}) = 1$ for given arguments $a_1, ..., a_{n_k} \in \{0, 1\}$ has been shown to be PSPACE-complete [6]. This immediately implies also the PSPACE-completeness of the NBF satisfiability problem, i.e. the problem of determining whether there exists a choice of arguments for which the defined function is true. This PSPACE-completeness suggests a close relationship to quantified Boolean formulas. In fact, it is not difficult to encode QBF formulas as NBFs. For example, let $\Phi(\mathbf{z}) = \forall x \exists y\, \phi(x, y, \mathbf{z})$ be a pQBF formula with matrix $\phi$ and free variables $\mathbf{z}$. First, we define the initial functions $f_\wedge(x_1, x_2) := x_1 \wedge x_2$, $f_\vee(x_1, x_2) := x_1 \vee x_2$ and $f_1(x, y, \mathbf{z}) := \phi$. Then we simulate the existential quantifier by means of the $\vee$ function, using the equivalence $\exists y\, \phi(x, y, \mathbf{z}) \approx \phi(x, 0, \mathbf{z}) \vee \phi(x, 1, \mathbf{z})$. This leads to the definition $f_2(x, \mathbf{z}) := f_\vee(f_1(x, 0, \mathbf{z}), f_1(x, 1, \mathbf{z}))$, and analogously $f_3(\mathbf{z}) := f_\wedge(f_2(0, \mathbf{z}), f_2(1, \mathbf{z}))$ to simulate the universal quantifier by the $\wedge$ function. By construction, we obtain $f_3(\mathbf{z}) \approx \Phi(\mathbf{z})$. Because of the need to count the variables in the arguments, the length of the resulting definition is in general not linear in the length of the formula, but only v-linear.

The inverse direction from NBF to QBF is less intuitive. By a general argument, every Boolean function defined by a NBF can be simulated by a poly-space Turing machine [6]. And a poly-space Turing machine can be encoded as a QBF of polynomial length [12]. Recently, a linear-time equivalence-preserving transformation from NBF to pQBF has been found [4].

**Lemma 1.** NBF $\leq_{linear}$ pQBF, QBF *and* pQBF, QBF $\leq_{v\text{-}linear}$ NBF.

Whether there exists a linear-time transformation from (p)QBF to NBF is an open question and closely related to the question whether for circuits there is a linear-time transformation to NBF.

## 5 Quantified Circuits

Similar to quantified Boolean formulas being an extension of propositional formulas, we now introduce *quantified circuits* as Boolean circuits which may in addition contain nodes labeled with $\forall x$ or $\exists x$ for a propositional variable $x$. Each of these *quantifier nodes* has exactly one incoming edge and an arbitrary number of outgoing edges. Examples of quantified circuits are given in Fig. 2. We say that a variable $x$ occurs *bound* in a quantified circuit if there is a path from the input $x$ or $\neg x$ to the sink which passes through a quantifier node $\exists x$ or $\forall x$. On the other hand, $x$ occurs *free* if there is a path from $x$ or $\neg x$ to the sink which contains no node $\exists x$ or $\forall x$. In the left circuit in Fig. 2, $x$ is bound, $z$ is free, and $y$ occurs free and bound.

We borrow the semantics of quantified circuits from quantified Boolean formulas by mapping to each edge of a quantified circuit a QBF formula in the following bottom-up manner: The formula for an input node $x$ or $\neg x$ is $x$ or $\neg x$ itself. The output edge of an AND-node over incoming edges associated with

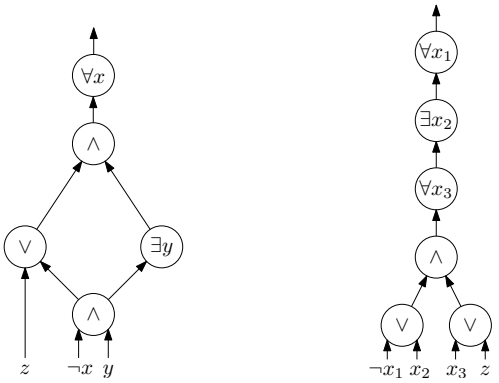


Fig. 2: Quantified circuits in non-prenex form (left) and prenex form (right)

formulas $\alpha(\mathbf{x})$ and $\beta(\mathbf{y})$ is then labeled with $\alpha(\mathbf{x}) \wedge \beta(\mathbf{y})$. OR- and NOT-nodes are treated analogously. When we encounter a quantifier node $\exists x$ that has an incoming edge labeled with $\alpha(x, \mathbf{y})$, we label its outgoing edges with $\exists x\, \alpha(x, \mathbf{y})$, and analogously $\forall x\, \alpha(x, \mathbf{y})$ for nodes $\forall x$. Finally, the interpretation of the circuit is defined as the value of the formula associated with its output when the variables are assigned as determined by the circuit inputs. For example, the left circuit in Fig. 2 is equivalent to $\forall x\ ((z \vee (\neg x \wedge y)) \wedge (\exists y(\neg x \wedge y)))$.

This semantics means that when we ignore the direction of the edges, a quantified circuit can be understood as the extension of a syntax tree of a quantified Boolean formula with additional sharing of subformulas. That is also the reason why we prefer to draw those circuits with the sink on the top.

When we construct the associated QBF formula as in the semantics definition, it is obvious that the length of the formula can be super-polynomial in the length (size) of the circuit, since we lose the sharing of subcircuits. It turns out that this can be avoided easily when we represent the quantified circuit as NBF. The encoding is the same as the one for ordinary Boolean circuits from Section 4, with the following extension: For an $\exists x$ node (a $\forall x$ node, respectively) that has an incoming edge labeled with $f(x, \mathbf{y})$, we define a new function symbol $g(\mathbf{y}) := f_\vee(f(0, \mathbf{y}), f(1, \mathbf{y}))$ ($g(\mathbf{y}) := f_\wedge(f(0, \mathbf{y}), f(1, \mathbf{y}))$, respectively).

**Proposition 3.** Quantified Circuits $\leq_{v\text{-}linear}$ NBF

### 5.1 Normal Forms

For QBF, various normal forms are well known. We now define analogous normal forms for quantified circuits and analyze them in the following subsections.

**Definition 4.** *Let $c$ be a quantified circuit. Then we say:*

1. *$c$ is in* negation normal form (NNF) *if each inner node is either an AND-node, an OR-node or a quantifier node, and the inputs are variables $x$ or negated variables $\neg x$. That means only inputs can be negated.*

2. *c is in* prenex form *if every successor node of a quantifier node is a quantifier node, too. That means quantifiers are only allowed in front of the sink.*
3. *c is* pure *if no variable has both bound and free occurrences in c.*

The quantified circuits shown earlier in Fig. 2 are both in negation normal form. The circuit on the left is neither in prenex form nor pure, while the one on the right is in prenex form and thus also pure. Subsequently, we assume that all quantified circuits are in negation normal form, unless stated otherwise.

**Proposition 4.** *A quantified circuit c is in negation normal form (in prenex form, or pure, respectively) if and only if the associated* QBF *formula is in negation normal form (in prenex form, or pure, respectively).*

In QBF, it is easy to transform NNF formulas into equivalent pure or even prenex formulas by renaming and shifting of quantifiers. Consider the example $\Phi = \forall x\ ((z \vee (\neg x \wedge y)) \wedge (\exists y\ (\neg x \wedge y)))$ where $y$ occurs free and bound. Then $\Phi \approx \forall x\ ((z \vee (\neg x \wedge y)) \wedge (\exists u\ (\neg x \wedge u))) \approx \forall x \exists u\ ((z \vee (\neg x \wedge y)) \wedge (\neg x \wedge u))$. But such renamings are problematic for non-prenex quantified circuits due to shared subcircuits. Consider again the left circuit in Fig. 2. If we rename $\exists y$ into $\exists u$, the resulting term $(\neg x \wedge u)$ can no longer be shared with $(\neg x \wedge y)$, which we still need to keep, because the free occurrence of $y$ cannot be renamed without losing equivalence. Thus, we need to have two copies of the previously shared subcircuit. In general, this can cause exponential growth. That observation motivates separate investigations of prenex and non-prenex circuits.

### 5.2 Quantified Circuits in Prenex Form

**Definition 5.** *(Quantified Circuits in Prenex Form)*
1. $\mathrm{Q}\mathcal{C} := \{c \mid c$ *is a circuit over* $\wedge, \vee, \neg, \forall$ *and* $\exists$ *in* NNF *and prenex form*$\}$
2. $\exists\mathcal{C} := \{c \mid c$ *is a circuit over* $\wedge, \vee, \neg$ *and* $\exists$ *in* NNF *and prenex form*$\}$

Because of the previously mentioned linear transformation of Boolean circuits into existentially quantified Boolean formulas by labeling edges with auxiliary variables as in Section 2, we immediately get the following relationships:

**Proposition 5.** $\exists\mathcal{C} =_{linear} \exists\mathrm{BF}$ *and* $\mathrm{Q}\mathcal{C} =_{linear} \mathrm{QBF}, \mathrm{pQBF}$.

### 5.3 Pure Quantified Circuits in Non-Prenex Form

Now, we investigate quantified circuits which are not in prenex form, but still pure. In the next section, we will also drop the purity restriction.

**Definition 6.** *(Pure Quantified Circuits in Negation Normal Form)*
1. $\mathcal{C}_{\forall,\exists} := \{c \mid c$ *is a pure circuit over* $\wedge, \vee, \neg, \forall$ *and* $\exists$ *in* NNF$\}$
2. $\mathcal{C}_{\exists} := \{c \mid c$ *is a pure circuit over* $\wedge, \vee, \neg$ *and* $\exists$ *in* NNF$\}$

In order to discuss the expressive power of $\mathcal{C}_{\exists}$, we further refine this class. The idea is to restrict the number of occurrences of $\exists x$ nodes for a single variable $x$.

**Definition 7.** *For $k \geq 1$, we let*
$\mathcal{C}_{\exists(k)} := \{c \in \mathcal{C}_\exists \mid$ *each variable occurs in at most $k$ $\exists$ nodes of the pure circuit $c\}$*

When $k = 1$, all $\exists$ nodes have distinct variable names. Furthermore, no variable occurs both free and bound, since the circuits are pure. That allows us to move all $\exists$ nodes in front of the sink, just like prenexing in QBF. As the circuits are also in NNF, we obtain an equivalent circuit in $\exists\mathcal{C}$. In the other direction, every $\exists\mathcal{C}$ circuit is trivially equivalent to a $\mathcal{C}_{\exists(1)}$ circuit, because if a prenex circuit has quantifier nodes with duplicate names, all but the innermost can be dropped.

**Proposition 6.** $\mathcal{C}_{\exists(1)} =_{linear} \exists\mathcal{C}$

For $k = 2$, we shall see that the expressive power jumps to the full power of quantified circuits. That means $\mathcal{C}_{\exists(2)}$ circuits can compactly encode universal quantifiers using only existential quantifiers. This is quite simple if we waive the NNF requirement and use the equivalence $\neg\exists x\phi \approx \forall x\neg\phi$. But if we only consider circuits and QBF formulas in NNF, this is not possible. In fact, a poly-time simulation of universal quantifiers in negation normal form QBF formulas would lead to the collapse of the polynomial hierarchy. Accordingly, the following idea is specific to circuits and uses structure sharing by fanout: It is well known that quantifiers can be expanded, e.g. by the equivalence $\forall x\, \alpha(x, \mathbf{y}) \approx \alpha(0, \mathbf{y}) \wedge \alpha(1, \mathbf{y})$ with some free variables $\mathbf{y}$. Repeated application clearly causes exponential growth due to the duplication of $\alpha$. Unfortunately, sharing by fanout only works for subexpressions which are exactly the same, but $\alpha$ has different arguments here. Our trick is to use the equivalence $\forall x\, \alpha(x, \mathbf{y}) \approx (\exists x(x \wedge \alpha(x, \mathbf{y})) \wedge (\exists x(\neg x \wedge \alpha(x, \mathbf{y}))$ instead. Now, we have two identical occurrences of $\alpha(x, \mathbf{y})$, which can be shared as shown in Fig. 3. By repeated expansion of universal nodes in this way, any quantified circuit in NNF can be transformed into a representation that uses only $\exists$ nodes. Obviously, the size of the resulting circuit remains linear in the size of the initial circuit.

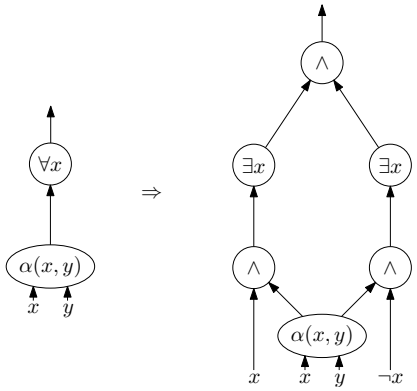


Fig. 3: ∀-simulation by ∃ nodes and sharing of subcircuits

We can now summarize our expressiveness results in the following theorem.

**Theorem 1.** $\mathcal{QC} =_p \mathcal{C}_{\forall,\exists} =_p \mathcal{C}_{\exists} =_p \mathcal{C}_{\exists(2)} =_p$ NBF $=_p$ QBF $=_p$ pQBF

*Proof.* We show QBF $\leq_p \mathcal{QC} \leq_p \mathcal{C}_{\exists(2)} \leq_p \mathcal{C}_{\exists} \leq_p \mathcal{C}_{\forall,\exists} \leq_p$ NBF $\leq_p$ QBF.

1. QBF $\leq_p \mathcal{QC}$ according to Proposition 5.
2. $\mathcal{QC} \leq_p \mathcal{C}_{\exists(2)}$ by encoding a universal node with two existentials (Fig. 3).
3. $\mathcal{C}_{\exists(2)} \leq_p \mathcal{C}_{\exists} \leq_p \mathcal{C}_{\forall,\exists}$: $\mathcal{C}_{\exists(2)}$ is a subset of $\mathcal{C}_{\exists}$, which in turn is a subset of $\mathcal{C}_{\forall,\exists}$.
4. $\mathcal{C}_{\forall,\exists} \leq_p$ NBF follows from Proposition 3.
5. NBF $\leq_p$ QBF by Lemma 1. □

### 5.4 Non-Pure Quantified Circuits

We now consider quantified circuits in which a variable may occur both free and bound. In the analogous QBF case, we can rename such bound variables in consideration of their scope, so that finally no variable is both free and bound. For example, $(\exists x\, \alpha(x,y)) \wedge \alpha(x,y) \approx (\exists x'\, \alpha(x',y)) \wedge \alpha(x,y)$. We have already pointed out that such renamings are problematic for circuits, since they make direct sharing of subcircuits impossible. In the example, $\alpha$ can no longer be shared, so we need two copies $\alpha(x',y)$ and $\alpha(x,y)$. Can we avoid such copying?

Clearly, non-pure circuits can be made pure by an indirect transformation: We know that all quantified circuits, including non-pure ones, can be encoded as v-linear NBFs, which in turn correspond to linear-size pure quantified circuits.

But there is also a direct linear-time transformation from non-pure to pure circuits: for every variable $x$ which has both free and bound occurrences in a given quantified circuit $c$, let $x'$ be a new variable which does not yet occur in $c$. Then we substitute $x'$ for all occurrences of $x$ in $c$, including occurrences in quantifier nodes. We call the resulting circuit $c[x/x']$. To make it equivalent to the original one, we bind $x'$ with an existential quantifier and require $x'$ to be true if and only if $x$ is true. We obtain a circuit which represents $\exists x'((x' = x) \wedge c[x/x'])$, as shown in Fig. 4 for the above example $(\exists x\, \alpha(x,y)) \wedge \alpha(x,y)$.

Essentially, this construction turns a free variable into a bound variable, so instead of having a free variable named like a bound variable, we now have bound variables with duplicate names (in Fig. 4, we get two quantifier nodes $\exists x'$). For the previously mentioned indirect transformation from non-pure to pure circuits via NBF, the situation is similar, because it requires universal quantifier nodes, and their simulation by existential ones as in Fig. 3 also introduces existential quantifier nodes with duplicate names. This is not unexpected, since pure circuits with distinct quantifier names, previously denoted $\mathcal{C}_{\exists(1)}$, seem to be significantly weaker ($\mathcal{C}_{\exists(1)} =_p \exists$BF) than those with duplicate names ($\mathcal{C}_{\exists(2)} =_p$ QBF). Interestingly, we can now show that the full expressiveness of QBF can be achieved by existentially quantified circuits with distinct quantifier names if we allow non-purity. Formally, the non-pure counterpart of $\mathcal{C}_{\exists(1)}$ is defined as follows:

**Definition 8.** $\mathcal{C}^*_{\exists(1)} := \{c \mid c$ *is a circuit over* $\wedge, \vee, \neg$ *and* $\exists$ *in negation normal form, every quantified variable occurs exactly once in the set of* $\exists$ *nodes* $\}$

In contrast to $\mathcal{C}_{\exists(1)} =_{linear} \exists$BF, we now prove $\mathcal{C}^*_{\exists(1)} =_{v-linear}$ QBF.

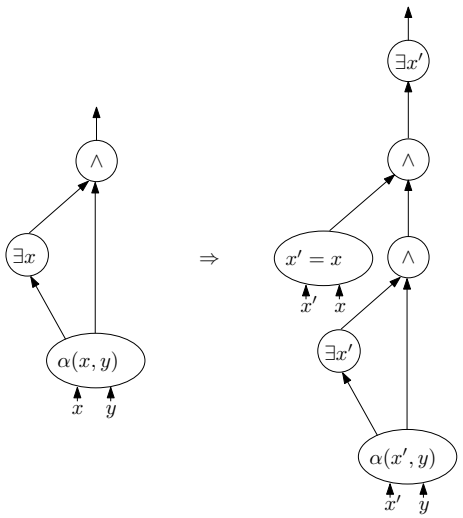


Fig. 4: Transformation of a non-pure into a pure circuit

**Theorem 2.** $\mathcal{C}^*_{\exists(1)} =_{v-linear}$ QBF

*Proof.* 1. $\mathcal{C}^*_{\exists(1)} \leq_{v-linear}$ QBF is obvious from the fact that any quantified circuit, and thus also a $\mathcal{C}^*_{\exists(1)}$ circuit, can be transformed into a v-linear NBF (Proposition 3), followed as before by a linear-time transformation to QBF.

2. We show QBF $\leq_{linear} \mathcal{C}^*_{\exists(1)}$ with a similar approach as for the transformation from QBF to $\mathcal{C}_{\exists(2)}$ from Theorem 1: we bring the formula into prenex form (with uniquely named variables) and use the obvious linear mapping from pQBF into a corresponding quantified circuit in prenex form. We then simulate the universal quantifier nodes using only existential nodes, but with a new procedure which is different from the one shown in Fig. 3.
The new procedure uses the equivalence $\forall x\ \alpha(x, \mathbf{y}) \approx \alpha(x, \mathbf{y}) \wedge \alpha(\neg x, \mathbf{y})$. Notice that $x$ is now free in the right-hand formula. Typically, equivalent formulas need to have the same free variables, but it is possible to have additional ones which occur only on one side, as long as their actual values do not influence the truth value of the formula. This is the case here with $x$: no matter whether $x = 0$ or $x = 1$, the formula on the right represents $\alpha(0, \mathbf{y}) \wedge \alpha(1, \mathbf{y})$, which is just the definition of universal quantification.
Now, we need to express this equivalence by a quantified circuit which contains only one copy of $\alpha$. Sharing of subcircuits by fanout only works if both instances of $\alpha$ have exactly the same arguments, say $\alpha(x, \mathbf{y})$. Our idea is to first introduce a new existential variable $x'$ which abbreviates $\neg x$:

$$\forall x\ \alpha(x, \mathbf{y}) \approx \alpha(x, \mathbf{y}) \wedge \exists x'((x' = \neg x) \wedge \alpha(x', \mathbf{y}))$$

In order to turn $\alpha(x', \mathbf{y})$ into $\alpha(x, \mathbf{y})$, we then perform another renaming, but this time, the new existential variable is named $x$, just like the free variable:

$$\forall x\ \alpha(x, \mathbf{y}) \approx \alpha(x, \mathbf{y}) \wedge \exists x'((x' = \neg x) \wedge \exists x((x = x') \wedge \alpha(x, \mathbf{y})))$$

We end up with two copies of $\alpha(x, \mathbf{y})$ that can be shared by fanout, as shown in Fig. 5. Everything else added to the resulting circuit has constant size, so repeated application of this procedure will lead to a circuit of linear size. It is clearly in $\mathcal{C}^*_{\exists(1)}$, since all quantified variables have distinct names. □

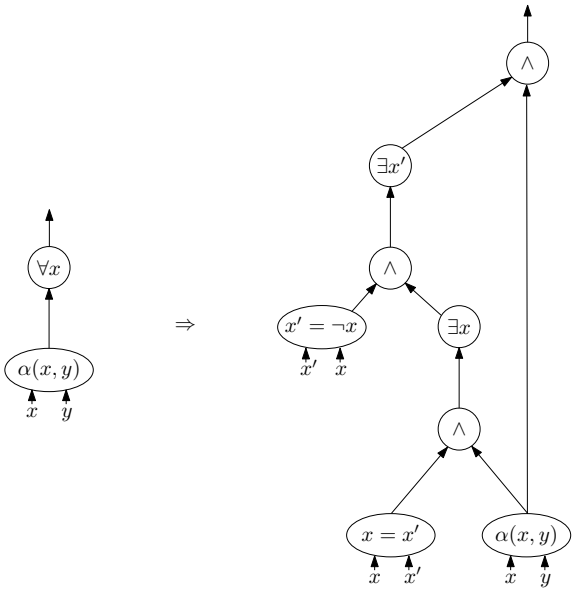


Fig. 5: ∀-simulation with unique quantifier names

We now have a second method to simulate universal quantifiers. Both of them require the ability to express overlapping scopes of variables with identical names, but the theorem shows that it does not matter whether that overlapping is between two bound variables or between a bound and a free variable.

The expressive power of $\mathcal{C}^*_{\exists(1)}$ is also evident from the following observation: In QBF, it is possible to compress conjunctions of formula instantiations for different variable names with the following equivalence [7]:

$$\alpha(x', y) \wedge \alpha(x, y) \approx \forall u\ (((u = x') \vee (u = x)) \rightarrow \alpha(u, y))$$

For the dual case $\alpha(x', y) \vee \alpha(x, y)$, we can use existential quantification:

$$\alpha(x', y) \vee \alpha(x, y) \approx \exists u\ (\alpha(u, y) \wedge ((u = x') \vee (u = x)))$$

In both cases, only one copy of $\alpha$ is needed. The second equivalence can obviously be applied exactly the same in $\mathcal{C}^*_{\exists(1)}$. An equivalence of the first form could be translated into $\mathcal{C}^*_{\exists(1)}$ by the above simulation of universal quantifiers, but it turns out that the same degree of compression can be achieved by a direct encoding without universal quantifiers. The idea is to introduce a local renaming of $x'$:

$$\alpha(x, y) \wedge \alpha(x', y) \approx \alpha(x, y) \wedge \exists x((x = x') \wedge \alpha(x, y))$$

Now, both instances of $\alpha(x, y)$ can be computed by one subcircuit (Fig. 6).

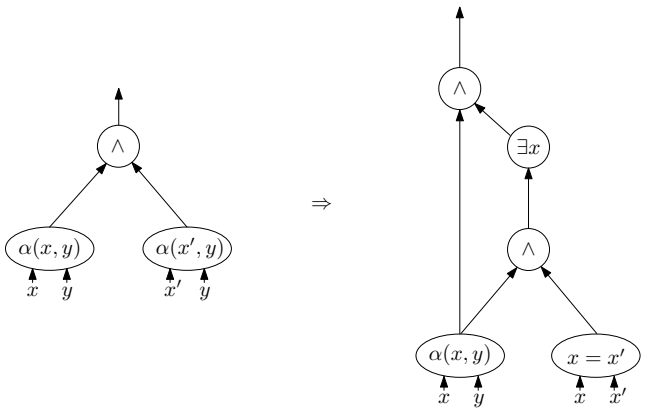


Fig. 6: Compression of renamed instantiations

## 6 Conclusion

We have studied quantified circuits, an extension of Boolean circuits with additional quantifier nodes. Such a circuit can be understood as syntax tree of a quantified Boolean formula with additional sharing of subformulas. In general, quantified circuits have the same expressive power under polynomial-time equivalence reductions as quantified Boolean formulas and nested Boolean functions:

$$\text{Quantified Circuits} =_p \text{QBF} =_p \text{NBF}$$

We could, however, prove the interesting result that every quantified circuit can be transformed into a polynomial-size equivalent circuit containing only existential quantifier nodes, which implies:

$$\mathcal{C}_\exists =_p \text{QBF}$$

This simulation of universal quantifiers does require the ability to express overlapping scopes of variables with identical names, be it pairs of bound variables or bound and free variables having the same names:

$$\mathcal{C}_{\exists(2)} =_p \mathcal{C}^*_{\exists(1)} =_p \text{QBF}$$

Such overlapping makes it possible to rename variables, and in combination with the subformula sharing ability of the underlying circuit structure allows compact encodings of subformula instantiations with different names, such as e.g. $\alpha(x, y) \wedge \alpha(x', y)$. With the construction from Fig. 6, this can be expressed with one copy of the circuit $\alpha(x, y)$ and without the need to use universal quantifiers as in the QBF encoding $\alpha(x', y) \wedge \alpha(x, y) \approx \forall u\ (((u = x') \vee (u = x)) \rightarrow \alpha(u, y))$. Ordinary Boolean circuits and purely existentially quantified Boolean formulas ($\exists$BF) seem to be lacking exactly that renaming ability and appear to be limited to abbreviate exact repetitions of subformulas. Nevertheless, it remains a long-standing open problem to show that QBF is indeed exponentially more powerful

than ∃BF. Perhaps, quantified circuits might provide a new perspective onto that problem, especially since they allow focusing only on one kind of quantifier and instead to consider renaming as the crucial feature.

## References


[1] S. Anderaa and E. Börger. *The Equivalence of Horn and Network Complexity for Boolean Functions.* Acta Informatica, 15:303–307, 1981.

[2] B. Borchert and F. Stephan. *Looking for an Analogue of Rice's Theorem in Circuit Complexity Theory.* Proc. 5th Kurt Gödel Colloquium (KGC 1997), Springer LNCS 1289, pp. 114–127, 1997.

[3] U. Bubeck and H. Kleine Büning. *Models and Quantifier Elimination for Quantified Horn Formulas.* Discrete Applied Mathematics, 156(10):1606–1622, 2008.

[4] U. Bubeck and H. Kleine Büning. *Encoding Nested Boolean Functions as Quantified Boolean Formulas.* 3rd Guangzhou Symposium on Satisfiability in Logic-based Modeling, Zhuhai, China, 2010.

[5] U. Bubeck and H. Kleine Büning. *Rewriting (Dependency-)Quantified 2-CNF with Arbitrary Free Literals into Existential 2-HORN.* Proc. 13th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2010), Springer LNCS 6175, pp. 58–70, 2010.

[6] S. Cook and M. Soltys. *Boolean Programs and Quantified Propositional Proof Systems.* The Bulletin of the Section of Logic, 28(3):119–129, 1999.

[7] N. Dershowitz, Z. Hanna, and J. Katz. *Bounded Model Checking with QBF.* Proc. 8th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2005), Springer LNCS 3569, pp. 408–414, 2005.

[8] U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda. *Comparing Different Prenexing Strategies for Quantified Boolean Formulas.* Proc. 6th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2003), Revised Selected Papers, Springer LNCS 2919, pp. 214–228, 2004.

[9] M. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik. *Combining Strengths of Circuit-based and CNF-based Algorithms for a High-performance SAT Solver.* Proc. 39th Design Automation Conference (DAC 2002), ACM, pp. 747–750, 2002.

[10] A. Goultiaeva and F. Bacchus. *Exploiting Circuit Representations in QBF Solving.* Proc. 13th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2010), Springer LNCS 6175, pp. 333–339, 2010.

[11] M. Kuehlmann, M. Ganai, and V. Paruthi. *Circuit-based Boolean Reasoning.* Proc. 38th Design Automation Conference (DAC 2001), ACM, pp. 232–237, 2001.

[12] A. Meyer and L. Stockmeyer. *Word Problems Requiring Exponential Time.* Proc. 5th ACM Symp. on Theory of Computing (STOC 1973), pp. 1–9, 1973.

[13] F. Pigorsch and C. Scholl. *Exploiting Structure in an AIG based QBF Solver.* Proc. Intl. Conf. on Design, Automation and Test in Europe (DATE 2009), pp. 1596–1601, 2009.

[14] P. Williams, A. Biere, E. Clarke, and A. Gupta. *Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking.* Proc. 12th Intl. Conf. on Computer Aided Verification (CAV 2000), Springer LNCS 1855, pp. 124–138, 2000.