Encoding Nested Boolean Functions as Quantified Boolean Formulas

Uwe Bubeck bubeck@upb.de
Hans Kleine Büning kbcsl@upb.de
Computer Science Institute,
University of Paderborn,
33098 Paderborn, Germany

Abstract

In this paper, we consider the problem of compactly representing nested instantiations of propositional subformulas with different arguments as quantified Boolean formulas (QBF). We develop a generic QBF encoding pattern which combines and generalizes existing QBF encoding techniques for simpler types of redundancy.

We obtain an equivalence-preserving transformation in linear time from the PSPACE-complete language of nested Boolean functions (NBF), also called Boolean programs, to prenex QBF. A transformation in the other direction from QBF to NBF is also possible in at most quadratic time by simulating quantifier expansion.

Keywords: *nested Boolean function, NBF, QBF, Boolean program, encoding*

Submitted May 2011; revised December 2011; published January 2012

1. Introduction

It has become quite popular to solve difficult decision problems by representing them as propositional formulas. Prominent examples include bounded model checking [2] and planning [6]. But for many interesting real-world problems, such encodings produce very large formulas, which makes it important to avoid redundancy. Quantified Boolean formulas (QBF) extend propositional logic with quantifiers over propositional variables, which often allows more compact encodings. While some problems have a natural forall-exists semantics which can elegantly be modeled by quantifiers [12], there also exist QBF encoding patterns which can further compress propositional problem representations. Well-known examples are the existentially quantified version of the Tseitin [14] and one-sided Tseitin [11] encoding and the sharing of transition relations in bounded model checking by universal quantification and iterative squaring [4, 5, 9]. These compression patterns, which we will later review in more detail, handle two basic sources of redundancy, namely exact repetitions of a subformula and the more general case of instantiating one subformula multiple times with different arguments (renaming). An example for the latter would be $\phi(A_1, B_1) \wedge \phi(A_2, B_2) \wedge \phi(A_3, B_3)$ with three instances of a propositional formula $\phi(u, v)$. By relatively simple existing quantifier patterns, such formulas can be represented with only one copy of ϕ.

In the following, we consider the generalization of this basic case to nested formula instantiations, as in the example $\alpha(\phi(A_1, B_1), \beta(\phi(A_2, B_2))) \wedge \beta(\phi(A_3, B_3))$ where the repeated instantiations of ϕ are in turn arguments to some other formulas α and β. Is it still possible to provide an equivalent QBF representation with only one copy of ϕ (and also only one of β) in a systematic way? The important difference between the two examples is that all three copies of ϕ must be true in the first example, whereas the second example contains no direct requirements on the truth of ϕ. Depending on the definition of α and β, ϕ might need to be true for some arguments, and false for others. As the main contribution of this paper, we are going to develop a generic QBF encoding pattern for such nested instantiations. The pattern will be formulated so that it immediately produces prenex QBF, sometimes denoted by pQBF. Considering that most QBF solvers still focus on prenex input, this saves an additional (linear-time) prenexing step.

Interestingly, the nested instantiation of propositional formulas is a powerful feature that has a similar expressiveness as quantification over propositional variables. A PSPACE-complete language can be obtained when allowing a set of initial functions represented as propositional formulas and compound functions which are defined as composition of previously defined or initial functions [3]. For example, let

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

be an initial function which computes the parity of two binary variables. Then the parity of four variables can be computed by reusing f_0:

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

The parity of 16 variables can be expressed compactly by reusing f_1, and so on. In [3], from which the example is taken, such function definitions are called a *Boolean program*, but that term is more prominently used for a different concept in the context of verification, so we prefer to speak of *Nested Boolean functions* (NBF).

As suggested informally in [8] and explained in Section 6 in more detail, it is easy to transform quantified Boolean formulas into equivalent nested Boolean functions with a linear number of function definitions by simulating quantifiers. With the QBF encoding pattern for nested formula instantiations that we develop in this paper, we obtain an equivalence-preserving transformation also in the other direction from NBF to QBF in linear time, which can be a valuable tool for proving expressiveness results. For example, we have been able to show in [8] close relationships between QBF and classes of quantified circuits (Boolean circuits with additional quantifier nodes not necessarily in prenex form), and the proofs of these results rely heavily on the NBF to QBF transformation that we are now going to establish ([8] anticipates that transformation without further details and references a preliminary workshop version of the paper at hand).

A further application of the transformation from NBF to QBF might be to encode (parts of) problems initially as nested Boolean functions by exploiting the intuitive nature of function composition, and then transform these NBF encodings in linear time into QBF to solve them with readily available QBF solvers [10]. That means nested Boolean functions might be used as an alternative input language for QBF solvers. A concrete example for modeling with NBF will be shown at the end of this paper, in Section 7.

A transformation from NBF to QBF seems also possible by adapting existing results [13] on BPLK and G, the proof systems based on the characterization of polynomial space as nested Boolean functions and quantified Boolean formulas, respectively. But in contrast to our linear transformation, the translation presented in [13] has quadratic length and produces non-prenex formulas of a rather complex structure, because the encoding of a function f_i contains the negated encoding of f_{i-1}.

2. Preliminaries

A *quantified Boolean formula* is a propositional formula or a formula of the form $\forall x\ \Phi(x)$ or $\exists y\ \Phi(y)$ where x, y are propositional variables and Φ is itself a quantified Boolean formula. *Universal quantification* $\forall x\ \Phi(x)$ is defined to be true if and only if $\Phi(0)$ is true *and* $\Phi(1)$ is true, and *existential quantification* $\exists y\ \Phi(y)$ means that $\Phi(0)$ *or* $\Phi(1)$ is true. An example of a syntactically correct QBF formula is $\forall x\,(\neg x \vee (\exists y\,(y \vee x \vee z)))$. For simplicity, we often consider formulas in *prenex form* $\Phi = Q_1 v_1 ... Q_k v_k\,\phi$ with quantifiers $Q_i \in \{\forall, \exists\}$ and a propositional formula ϕ. We call $Q := Q_1 v_1 ... Q_k v_k$ the *prefix* and ϕ the *matrix* of Φ. Prenex QBF formulas are sometimes denoted by pQBF, but we do not make that distinction here, since pQBF $\subseteq$ QBF and all our results that take a QBF formula as input are also valid for non-prenex QBF.

Variables on which a quantifier is applied are called *bound* variables, and variables which are not bound by a quantifier are *free*. In the last example, x and y are bound, and z is free. If a QBF formula contains free variables, we call it *open*, and *closed* otherwise. A closed QBF formula is either true or false. The reason is that if we did expand from left to right all quantifiers according to their above definitions $\forall x\ \Phi(x) := \Phi(0) \wedge \Phi(1)$ and $\exists y\ \Phi(y) := \Phi(0) \vee \Phi(1)$, we would obtain a propositional formula (of exponential size) that contains no variables, but only Boolean operators and truth values. Since being true coincides with being satisfiable for such formulas, we can use the terms "true" and "satisfiable" synonymously for closed QBF. In contrast, the truth value of an open QBF formula depends on the value of the free variables. 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 τ 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 τ for the free variables in Φ.

QBF 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*, written as $\Phi(z_1, ..., z_r) \approx \Psi(w_1, ..., w_s)$ or simply $\Phi \approx \Psi$, if and only if for every truth assignment τ to the free variables $z_1, ..., z_r, w_1, ..., w_s$ both formulas evaluate to the same truth value. When we consider propositional formulas as quantified Boolean formulas in which all variables are free, this definition of logical equivalence is simply a generalization of the usual equivalence criterion for propositional formulas. Accordingly, we can also consider logical equivalence between a propositional formula and a QBF formula with free variables that correspond to the variables in the propositional formula.

It is important to notice that bound variables are not directly considered when checking for logical equivalence, which makes these variables local to the respective formula. This makes it possible to add auxiliary variables to a formula, e.g. to abbreviate repeating parts as described in the next section, without losing logical equivalence. This is a powerful advantage of QBF over ordinary propositional calculus, because propositional formulas can usually only be logically equivalent if they have exactly the same variables. This problem can be avoided by considering only *satisfiability equivalence* $\approx_{SAT}$, which requires that if one of the formulas is satisfied by an assignment to its variables, the other one must also have some satisfying truth assignment (possibly of a different structure). 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.

The *length* of a quantified Boolean formula is the number of variable occurrences, including occurrences with quantifiers. For example, the formula $\forall x\,(\neg x \vee (\exists y\,(y \vee x \vee z)))$ has length 6. For the typically considered formulas in negation normal form (NNF), where a negation sign can only appear immediately in front of a variable, this length measure differs by at most a constant factor from the alternative definition of counting also propositional operators and quantifier symbols.

For the equivalence operator (i.e. the bidirectional implication), we use the symbol "$=$". That means $\sigma = \pi$ is $(\neg\sigma \vee \pi) \wedge (\sigma \vee \neg\pi)$ for formulas σ and π. For two tuples of variables $\mathbf{x} = (x_1, ..., x_k)$ and $\mathbf{y} = (y_1, ..., y_k)$, we let $\mathbf{x} = \mathbf{y}$ be an abbreviation for $\bigwedge_{i=1,...,n}(x_i = y_i) \approx \bigwedge_{i=1,...,n}((\neg x_i \vee y_i) \wedge (x_i \vee \neg y_i))$. To save parentheses, we assume that "$=$" has a higher binding priority than one-sided implication, so $\alpha = \beta \rightarrow \gamma$ means $(\alpha = \beta) \rightarrow \gamma$.

3. Basic QBF Encoding Patterns

The QBF encoding of nested Boolean functions that we develop in Section 5 can be seen as a clever combination of existing QBF encoding patterns. The first of these basic patterns is the well-known technique of using existentially quantified auxiliary variables to introduce abbreviations for exact repetitions of subformulas [14]. Consider the following example:

$$(A \vee \neg B \vee C \vee D) \wedge (A \vee \neg B \vee C \vee \neg E) \wedge (A \vee \neg B \vee C \vee F)$$

Here, $A \vee \neg B \vee C$ is repeated three times. By introducing a new variable y as an abbreviation for that term, we can get a logically equivalent formulation that contains only one copy of $A \vee \neg B \vee C$:

$$\exists y\ (y = (A \vee \neg B \vee C)) \wedge (y \vee D) \wedge (y \vee \neg E) \wedge (y \vee F) \tag{1}$$

If the abbreviated term occurs only in one polarity (which is always the case for non-atomic subformulas of input formulas in NNF), it is in fact sufficient to use a one-sided implication $y \rightarrow \psi$ to state that y is an abbreviation for exact occurrences of ψ [11]. That means we obtain the logically equivalent formula

$$\exists y\ (y \rightarrow (A \vee \neg B \vee C)) \wedge (y \vee D) \wedge (y \vee \neg E) \wedge (y \vee F)\ .$$

Propositional logic with existential quantification appears to be quite powerful. It is well known that there are existentially quantified Boolean formulas in conjunctive normal form (CNF) for which every logically equivalent propositional CNF formula is exponentially longer. Closely related to the idea of defining abbreviations for exact repetitions of subformulas is the concept of fan-out in Boolean circuits. In fact, it has been shown that Boolean circuits with arbitrary fan-out have the same expressive power as existentially quantified CNF formulas in which the bound variables satisfy the Horn property [1, 7].

Universal quantification has been found helpful to express renamings of variables. Consider the example

$$\phi(A_1, B_1) \wedge \phi(A_2, B_2) \wedge \phi(A_3, B_3)$$

where we have multiple instances of a propositional formula ϕ for different arguments. Such situations can be compressed by replacing these instantiations with a single one that has universally quantified variables as arguments, say $\phi(u, v)$. Whenever u and v represent a tuple of arguments in the original formula, $\phi(u, v)$ must be true [4, 9]:

$$\forall u \forall v \ \left(\bigvee_{i=1}^{3} ((u = A_i) \wedge (v = B_i)) \right) \rightarrow \phi(u, v) \tag{2}$$

As it is, this trick works only for conjunctions of multiple instantiations as in the example, but not for expressions such as $\phi(A_1, B_1) \rightarrow \phi(A_2, B_2)$. The encoding that we present in Section 5 generalizes the idea to arbitrary enclosing formulas, e.g. expressions of the form $\psi(\phi(A_1, B_1), \phi(A_2, B_2))$ for arbitrary ψ.

Unless the polynomial hierarchy collapses, the full expressive power of quantified Boolean formulas can only be obtained by encodings which combine existential and universal quantification. A well-known approach which uses both kinds of quantifiers is non-copying iterative squaring [9], which can be considered as a special case of compressing a conjunction of renamed instances where the second argument of one instantiation is also the first argument of the next instantiation. Consider the following example:

$$\Phi(x_0, x_n) := \exists x_1 ... \exists x_{n-1} \ (\phi(x_0, x_1) \wedge \phi(x_1, x_2) \wedge ... \wedge \phi(x_{n-2}, x_{n-1}) \wedge \phi(x_{n-1}, x_n))$$

Expressions of this form occur, e.g., when we have a graph given by a transition relation ϕ and we want to express that vertices x_0 and x_n are connected by a continuous path. The idea of iterative squaring is to take the vertex y in the middle and reduce the problem to finding two paths of half the length:

$$\Phi_{2^i}(x_0, x_n) := \exists y \ (\Phi_{2^{i-1}}(x_0, y) \wedge \Phi_{2^{i-1}}(y, x_n))$$

Then the previous pattern can be used to compress the two instantiations of Φ_i into a single one with universally quantified arguments:

$$\Phi_{2^i}(x_0, x_n) := \exists y \forall u \forall v \ [(((u, v) = (x_0, y)) \vee ((u, v) = (y, x_n))) \rightarrow \Phi_{2^{i-1}}(u, v)]$$

From the classical proof of the PSPACE-hardness of QBF [9], it is well known that non-copying iterative squaring can encode any computation of a polynomial-space Turing machine into a polynomial-size QBF formula. The encoding pattern that we are going to develop in this paper is just as powerful, but based on a different and perhaps more intuitive characterization of polynomial space by nested Boolean functions.

4. Nested Boolean Functions

As mentioned in the introduction, nested Boolean functions are defined by composition of previously defined or initial functions, the latter being given as propositional formulas.

Definition 1. *(Nested Boolean Function)*
A nested Boolean function (NBF) *is a finite sequence $D(f_k) = (f_0, ..., f_k)$ of Boolean functions. For fixed $t \in \{0, ..., k\}$, a* NBF *consists of*

- initial functions $f_0, ..., f_t$, *which are each defined by* $f_i(\mathbf{x}^i) := \alpha_i(\mathbf{x}^i)$ *for a propositional formula* α_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 the Boolean constants* 0 *and* 1, *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}.

The argument variables are considered unique for each function definition, that means all variables in $\mathbf{x}^i$ *are different from the ones in* $\mathbf{x}^j$ *for any* i, j *with* $i \neq j$. *We call* f_k *the* defined Boolean function. *The length of a nested Boolean function* $D(f_k) = (f_0, ..., f_k)$ *is* $|D(f_k)| := |f_0| + ... + |f_k|$, *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 .

Consider again the parity example from the introduction:

$$\begin{aligned} f_0(p_1, p_2) &:= (\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2) \\ f_1(p_1, p_2, p_3, p_4) &:= f_0(f_0(p_1, p_2), f_0(p_3, p_4)) \end{aligned}$$

Then $D(f_1) = (f_0, f_1)$ has length $4 + 7$. For technical reasons, the definition requires the argument variables to be unique. We can still use the same names for arguments in different functions, but we must keep in mind that they denote different objects. That means we should understand the above parity example as

$$\begin{aligned} f_0(p_1, p_2) &:= (\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2) \\ f_1(q_1, q_2, q_3, q_4) &:= f_0(f_0(q_1, q_2), f_0(q_3, q_4)) \,. \end{aligned}$$

Using the notation of the previous definition, f_1 now has arguments $\mathbf{x}^1 = (q_1, q_2, q_3, q_4)$. With $\mathbf{x}^1_i$, we denote those variables of $\mathbf{x}^1$ which are passed on to called functions in the definition of f_1. Here, $\mathbf{x}^1_1 = (q_1, q_2)$ and $\mathbf{x}^1_2 = (q_3, q_4)$ (these do not have to be disjoint in general).

The semantics of NBF is now defined by a fallback to propositional logic. By iterated substitution, we can construct for each function in a NBF an associated propositional formula, called the *defined formula*. For an initial function, this is the right-hand side of its defining equation. Then we construct defined formulas for the compound functions $f_{t+1}, ..., f_k$ in this order. For a definition of the form $f_i(\mathbf{x}^i) := f_{j_0}(f_{j_1}(\mathbf{x}^i_1), ..., f_{j_r}(\mathbf{x}^i_r))$, we treat the defined formulas of the functions $f_{j_0}, ..., f_{j_r} \in \{f_0, ..., f_{i-1}\}$ as formula schemes with scheme variables that are replaced by formulas according to the caller's arguments.

Definition 2. *(*NBF *Semantics)*
We associate with each function $f_i \in (f_0, ..., f_k)$ *in a* NBF *a* defined formula $\mathit{def}(f_i)$. *For an initial function* $f_i(\mathbf{x}^i) := \alpha_i(\mathbf{x}^i)$, *this is simply* $\mathit{def}(f_i) := \alpha_i$.
The case of a compound function requires some additional notation: for a propositional formula ϕ, *let* $\phi[v_1/\sigma_1, ..., v_m/\sigma_m]$ *be the (simultaneous) substitution of* σ_i *for all occurrences of a variable* v_i *in* ϕ *for all* $i = 1, ..., m$. *We also write* $\phi[\mathbf{v}/\sigma]$ *for tuples* $\mathbf{v} = (v_1, ..., v_m)$ *and* $\sigma = (\sigma_1, ..., \sigma_m)$.
Then we associate a compound function of the form $f_i(\mathbf{x}^i) := f_{j_0}(f_{j_1}(\mathbf{x}^i_1), ..., f_{j_r}(\mathbf{x}^i_r))$ *with*

$$\mathit{def}(f_i) := \mathit{def}(f_{j_0})[x^{j_0,1}/\mathit{def}(f_{j_1})[\mathbf{x}^{j_1}/\mathbf{x}^i_1], ..., x^{j_0,r}/\mathit{def}(f_{j_r})[\mathbf{x}^{j_r}/\mathbf{x}^i_r]] \,.$$

Here, the subscripts $j_0, ..., j_r$ indicate the indices of the functions that are called in the definition of f_i, and $\mathbf{x}^{j_0}, ..., \mathbf{x}^{j_r}$ are the arguments of those functions.
We let the interpretation of a NBF $D(f_k) = (f_0, ..., f_k)$ *be the interpretation of the defined formula* $def(f_k)$.

For the preceding parity example

$$\begin{aligned} f_0(p_1, p_2) &:= (\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2) \\ f_1(p_1, p_2, p_3, p_4) &:= f_0(f_0(p_1, p_2), f_0(p_3, p_4)) \end{aligned}$$

we obtain:

$$\begin{aligned} def(f_0) &:= (\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2) \\ def(f_1) &:= def(f_0)[p_1/\left((\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2)\right), p_2/\left((\neg p_3 \wedge p_4) \vee (p_3 \wedge \neg p_4)\right)] \\ &= (\neg\left((\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2)\right) \wedge \left((\neg p_3 \wedge p_4) \vee (p_3 \wedge \neg p_4)\right)) \\ &\quad \vee (\left((\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2)\right) \wedge \neg\left((\neg p_3 \wedge p_4) \vee (p_3 \wedge \neg p_4)\right)) \end{aligned}$$

The example nicely illustrates the rapid growth from iterated substitution: $def(f_0)$ has only length 4, but $def(f_1)$ has already length 16. In general, the defined formula $def(f_k)$ of a NBF $D(f_k)$ can have exponential length. The main contribution of this paper is to show that for every nested Boolean function $D(f_k) = (f_0, ..., f_k)$, there exists a linear-size QBF Φ which is logically equivalent to $def(f_k)$, that means $\Phi \approx def(f_k)$. In the following, we will simply write $\Phi \approx f_k$ and use f_i and $def(f_i)$ interchangeably.

Please notice that for a NBF $D(f_k) = (f_0, ..., f_k)$ and given arguments $a_1, ..., a_{n_k} \in \{0, 1\}$, we can evaluate whether $f_k(a_1, ..., a_{n_k}) = 1$ (that means $def(f_k)[x^{k,1}/a_1, ..., x^{k,n_k}/a_{n_k}] = 1$) without constructing the whole defined formula $def(f_k)$. By immediately replacing subterms whenever their values are known, the formula can be simplified on-the-fly, and polynomial space is sufficient. In fact, the problem of evaluating a NBF has been shown to be PSPACE-complete [3]. 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 formula is true.

5. Transformation from NBF to QBF

We now develop our transformation from NBF to QBF by considering the simple example $f(x_1, x_2) := \psi(\phi(x_1), \phi(x_2))$ with two instances of ϕ for arguments x_1 and x_2. Our goal is to gradually construct a logically equivalent formula that contains only one instance of ϕ.

We begin by introducing abbreviations b^1 and b^2 for $\phi(x_1)$ and $\phi(x_2)$, respectively. This is achieved by existentially quantified auxiliary variables as in the first pattern (Eq. 1) from Section 3:

$$\exists b^1 \exists b^2 \; (\psi(b^1, b^2) \wedge (b^1 = \phi(x_1)) \wedge (b^2 = \phi(x_2)))$$

Now, we want to merge $\phi(x_1)$ and $\phi(x_2)$. The second pattern (Eq. 2) from Section 3 shows us how to compress a formula of the form $\phi(x_1) \wedge \phi(x_2)$ into a universally quantified formula $\forall x \; (((x = x_1) \vee (x = x_2)) \rightarrow \phi(x))$, but the formula at hand has a different structure $(b^1 = \phi(x_1)) \wedge (b^2 = \phi(x_2))$. Our idea is to use the previously mentioned pattern only in its special case $\psi(x_1) \approx \forall x \; ((x = x_1) \rightarrow \psi(x))$ with $\psi(x_1) := (b^1 = \phi(x_1))$, and analogously for $b^2 = \phi(x_2)$. We can use the same universal variable in both cases, because $(\forall u\, \alpha(u)) \wedge (\forall v\, \beta(v)) \approx \forall u\ (\alpha(u) \wedge \beta(u))$.

$$\exists b^1 \exists b^2 \forall x\ (\psi(b^1, b^2) \wedge (x = x_1 \to b^1 = \phi(x)) \wedge (x = x_2 \to b^2 = \phi(x)))$$

Using the pattern separately on single instantiations seems counter-productive in terms of formula length, but we now have two occurrences of $\phi(x)$ on which we can again apply the pattern for abbreviating exact repetitions by an existential variable (Eq. 1 from Section 3):

$$\exists b^1 \exists b^2 \forall x \exists y\ (\psi(b^1, b^2) \wedge (x = x_1 \to b^1 = y) \wedge (x = x_2 \to b^2 = y) \wedge (y = \phi(x)))$$

The resulting formula contains only one copy of ϕ and is logically equivalent to the original formula $f(x_1, x_2)$, due to the correctness of the individual compression patterns from Section 3. A detailed equivalence proof for our transformation will be given later (Theorem 1). Please notice that in each of the three steps of this transformation, the newly added quantifier(s) must be placed to the right of all previously introduced quantifiers, because the new variable(s) must depend on all variables already present in the formula. To be more precise: let $\Phi = Q\phi$ be a prenex QBF formula with prefix Q and matrix ϕ. If we find a formula $\Psi = Q'\psi \in$ QBF which is logically equivalent to the matrix of Φ, that means $\phi \approx Q'\psi$, we also have $\Phi \approx QQ'\psi$ (this can be seen as temporarily treating the variables in Q as free variables).

In a more complex scenario with multiple nested function definitions, the function f from our example might itself occur multiple times with different arguments, e.g. in a definition $g(x_1, ..., x_4) := \gamma(f(x_1, x_2), f(x_3, x_4))$. Such situations can be handled by applying the previously presented steps recursively. That means we first use the technique to create one single instance of f, say $f(u, v)$, and then we replace that instance with its definition $\psi(\phi(u), \phi(v))$ and proceed analogously with ϕ. In case that ϕ occurs also in the definition of other functions from the same NBF, it is important for avoiding exponential growth that we do not introduce multiple abbreviations of the form $\forall \mathbf{x} \exists y\ (y = \phi(\mathbf{x}))$. Instead, we associate with every $f_i \in (f_0, ..., f_k)$ one uniquely named abbreviation $y^i = f_i(\mathbf{x}^i)$ which is used for all instantiations of f_i in the NBF. Please remember that when we write $y^i = f_i(\mathbf{x}^i)$, we actually mean $y^i = \mathit{def}(f_i)$ as pointed out earlier. The complete encoding is given in the following definition:

Definition 3. *(Transformation from* NBF *to* QBF*)*
Let $D(f_k) = (f_0, ..., f_k)$ be a nested Boolean function. Then we map $f_0, ..., f_k$ to a set of prenex QBF *formulas $\Phi_i(\mathbf{x}^i) = Q_i\, \phi_i$. Q_i denotes the prefix and ϕ_i the matrix of the i-th formula for all $i = 0, ..., k$.*
For the initial functions $f_0, ..., f_t$ which are defined by propositional formulas $f_i(\mathbf{x}^i) := \alpha_i(\mathbf{x}^i)$ over variables $\mathbf{x}^i$, we obtain:

$$\Phi_0(\mathbf{x}^0) := \exists y^0\ (y^0 = f_0(\mathbf{x}^0))$$

$$\Phi_i(\mathbf{x}^i) := \exists y^i \forall \mathbf{x}^{i-1} Q_{i-1}\ (\phi_{i-1}(\mathbf{x}^{i-1}, ..., \mathbf{x}^0, y^{i-1}, ..., y^0) \wedge (y^i = f_i(\mathbf{x}^i))),\ i = 1, ..., t$$

For compound functions $f_i(\mathbf{x}^i) := f_{j_0}(f_{j_1}(\mathbf{x}^i_1), ..., f_{j_r}(\mathbf{x}^i_r))$*,* $i = t+1, ..., k$*, we construct* QBF *formulas of the following form:*

$$\begin{aligned}\Phi_i(\mathbf{x}^i) \quad := \quad & \exists y^i \exists b^{i,1} ... \exists b^{i,r} \forall \mathbf{x}^{i-1} Q_{i-1} \\ & (\phi_{i-1}(\mathbf{b}^{i-1}, ..., \mathbf{b}^{t+1}, \mathbf{x}^{i-1}, ..., \mathbf{x}^0, y^{i-1}, ..., y^0) \wedge \\ & (\mathbf{x}^i_1 = \mathbf{x}^{j_1} \rightarrow b^{i,1} = y^{j_1}) \wedge ... \wedge (\mathbf{x}^i_r = \mathbf{x}^{j_r} \rightarrow b^{i,r} = y^{j_r}) \wedge \\ & ((b^{i,1}, ..., b^{i,r}) = \mathbf{x}^{j_0} \rightarrow y^i = y^{j_0}))\end{aligned}$$

Finally, we let $\Phi(\mathbf{x}^k) := Q_k\ (\phi_k \wedge y^k)$ *be the* QBF *encoding of* $D(f_k)$*.*

Consider a simple example:
Let $f_0(x^{0,1}, x^{0,2}) := x^{0,1} \wedge x^{0,2}$ and $f_1(x^{1,1}) := \neg x^{1,1}$ be initial functions, and let
$f_2(x^{2,1}, x^{2,2}) := f_1(f_0(x^{2,1}, x^{2,2}))$ be a compound function.
Then we obtain the following QBF formulas:

$$\begin{aligned}\Phi_0(\mathbf{x}^0) \quad := \quad & \exists y^0\ (y^0 = (x^{0,1} \wedge x^{0,2})) \\ \Phi_1(\mathbf{x}^1) \quad := \quad & \exists y^1 \forall \mathbf{x}^0 \exists y^0\ ((y^0 = (x^{0,1} \wedge x^{0,2})) \wedge (y^1 = \neg x^{1,1})) \\ \Phi_2(\mathbf{x}^2) \quad := \quad & \exists y^2 \exists b^{2,1} \forall \mathbf{x}^1 \exists y^1 \forall \mathbf{x}^0 \exists y^0\ ((y^0 = (x^{0,1} \wedge x^{0,2})) \wedge (y^1 = \neg x^{1,1}) \\ & \wedge ((x^{2,1}, x^{2,2}) = (x^{0,1}, x^{0,2}) \rightarrow b^{2,1} = y^0) \wedge (b^{2,1} = x^{1,1} \rightarrow y^2 = y^1))\end{aligned}$$

Notice that $\Phi_0, ..., \Phi_2$ are all tautological, because Φ_i essentially says that there exists an abbreviation y^i for $f_i(\mathbf{x}^i)$, which is of course true for each choice of arguments $\mathbf{x}^i$. To obtain a formula that is true whenever f_2 is true, we need to add y^2 as a unit clause. We will now formally prove the equivalence of the resulting formula and the nested Boolean function.

Theorem 1. *Let* $D(f_k) = (f_0, ..., f_k)$ *be a nested Boolean function, and let* Φ *be its* QBF *encoding according to Definition 3.*
Then $f_k(\mathbf{x}^k) \approx \Phi(\mathbf{x}^k)$*, and the length of* $\Phi(\mathbf{x}^k)$ *and the time to construct it are both linear in the length of* $(f_0, ..., f_k)$*.*

Proof: The linearity is obvious: for each initial function, we add universal quantifiers $\forall \mathbf{x}^i$ of size at most $|f_i(\mathbf{x}^i)|$ and one existentially quantified variable that occurs once in the corresponding formula. In the case of a compound function, notice that $\mathbf{x}^i_1, ..., \mathbf{x}^i_r$ are already contained in the definition of the function, so $|(\mathbf{x}^i_1 = \mathbf{x}^{j_1} \rightarrow b^{i,1} = y^{j_1}) \wedge ... \wedge (\mathbf{x}^i_r = \mathbf{x}^{j_r} \rightarrow b^{i,r} = y^{j_r})| \leq 4 \cdot |f_i(\mathbf{x}^i)|$.
To prove the correctness of this encoding, we now show by induction on i that for all $\Phi_i(\mathbf{x}^i) = Q_i\ \phi_i(\mathbf{b}^i, ..., \mathbf{b}^{t+1}, \mathbf{x}^i, ..., \mathbf{x}^0, y^i, ..., y^0)$ the following holds: $\forall \mathbf{x}^i Q_i\ \phi_i$ is true and $Q_i\ (\phi_i \wedge \neg(y^i = f_i(\mathbf{x}^i)))$ is false for all truth assignments τ to $\mathbf{x}^i$. The combination of these two properties means that, for any τ and i, there is exactly one satisfying choice for y^i, namely $y^i := f_i(\tau(\mathbf{x}^i))$. This implies $f_i(\mathbf{x}^i) \approx Q_i\ (\phi_i \wedge y^i)$, and thus the desired equivalence $f_k(\mathbf{x}^k) \approx \Phi(\mathbf{x}^k)$.
For $i = 0$, both properties are obviously true, and for the remaining initial functions, i.e. $i = 1, ..., t$ (if $t > 0$), the claim clearly holds as well, because $Q_i\ \phi_i \approx (\forall \mathbf{x}^{i-1} Q_{i-1} \phi_{i-1}) \wedge \exists y^i (y^i = f_i(\mathbf{x}^i))$ and the first part of this formula is true by the induction hypothesis. On the other hand, $Q_i\ (\phi_i \wedge \neg(y^i = f_i(\mathbf{x}^i))) \approx (\forall \mathbf{x}^{i-1} Q_{i-1} \phi_{i-1}) \wedge \exists y^i ((y^i = f_i(\mathbf{x}^i)) \wedge \neg(y^i = f_i(\mathbf{x}^i)))$

is trivially unsatisfiable.
For the compound functions, that is $i = t+1, ..., k$, consider in Φ_i the clauses $(\mathbf{x}^i_1 = \mathbf{x}^{j_1} \to b^{i,1} = y^{j_1}) \wedge ... \wedge (\mathbf{x}^i_r = \mathbf{x}^{j_r} \to b^{i,r} = y^{j_r})$. Since $\mathbf{x}^{j_1}, ..., \mathbf{x}^{j_r}$ are universally quantified, these clauses are satisfied for all values of the universal variables if and only if $b^{i,l}$ is chosen so that $b^{i,l} = y^{j_l}$ whenever $\mathbf{x}^i_l = \mathbf{x}^{j_l}$, $l = 1, ..., r$. Now notice that $\forall \mathbf{x}^{j_l} Q_{j_l} \phi_{j_l}$ is a subformula of Φ_i, and that this subformula is true due to the first part of the induction hypothesis. Then the second part of the induction hypothesis implies that for all values of $\mathbf{x}^{j_l}$, we must have $y^{j_l} = f_{j_l}(\mathbf{x}^{j_l})$. In total, that means the clauses $(\mathbf{x}^i_l = \mathbf{x}^{j_l} \to b^{i,l} = y^{j_l})$ in Φ_i are satisfied if and only if $b^{i,l} = f_{j_l}(\mathbf{x}^{j_l})$ whenever $\mathbf{x}^i_l = \mathbf{x}^{j_l}$, and thus if and only if $b^{i,l} = f_{j_l}(\mathbf{x}^i_l)$. With $b^{i,l}$ being quantified in the outermost quantifier block of Φ_i, it is always possible to assign $b^{i,l}$ so that $b^{i,l} = f_{j_l}(\mathbf{x}^i_l)$. An analogous argument can now be applied to the last clause $((b^{i,1}, ..., b^{i,r}) = \mathbf{x}^{j_0} \to y^i = y^{j_0})$, and it follows that y^i must be chosen so that $y^i = f_{j_0}(b^{i,1}, ..., b^{i,r}) = f_{j_0}(f_{j_1}(\mathbf{x}^i_1), ..., f_{j_r}(\mathbf{x}^i_r)) = f_i(\mathbf{x}^i)$. Again, this choice is always possible, since $\exists y^i$ is the outermost quantifier. Together with the induction hypothesis that $\forall \mathbf{x}^{i-1} Q_{i-1}\, \phi_{i-1}$ is true, we have now shown that all clauses of Φ_i are satisfied for all values of $\mathbf{x}^i$ (which implies the first part of the induction claim), and that we must choose y^i so that $y^i = f_i(\mathbf{x}^i)$, which proves the second part of the claim. $\square$

There are several possible variations of our encoding. For example, it may be better in practice to copy relatively simple initial functions instead of abbreviating them. That means when we have a compound definition of the form $f_i(\mathbf{x}^i) := f_{j_0}(f_{j_1}(\mathbf{x}^i_1), ..., f_{j_r}(\mathbf{x}^i_r))$ and $f_{j_1}, ..., f_{j_r}$ are complex compound or initial functions, but f_{j_0} is simple, we can replace $((b^{i,1}, ..., b^{i,r}) = \mathbf{x}^{j_0} \to y^i = y^{j_0})$ in the original encoding with $(y^i = f_{j_0}(b^{i,1}, ..., b^{i,r}))$, that means we directly apply f_{j_0} on the intermediate results. For example, if $f_{j_0}(x^{j_0,1}, x^{j_0,2}) := x^{j_0,1} \to x^{j_0,2}$, we would simply write $y^i = (b^{i,1} \to b^{i,2})$. If we use copying in all places where f_{j_0} occurs, there is no need to introduce an abbreviation of the form $\forall \mathbf{x}^{j_0} \exists y^{j_0}\ (y^{j_0} = f_{j_0}(\mathbf{x}^{j_0}))$ in the first place.

Another variation of our encoding is to allow quantifiers in the definition of Boolean functions. When we allow the initial functions to be QBF instead of propositional formulas, the transformation remains as it is, since an expression of the form $\forall \mathbf{x}^i \exists y^i\ (y^i = f_i(\mathbf{x}^i))$ behaves no differently with respect to $\mathbf{x}^i$ and y^i if f_i is represented not as a propositional formula, but as a QBF formula with free variables $\mathbf{x}^i$. For compound functions with quantifiers, e.g. $f_i(\mathbf{x}^i) := \exists v\ f_{j_0}(f_{j_1}(v, \mathbf{x}^i_1), ..., f_{j_r}(v, \mathbf{x}^i_r))$, we have the problem that the previously presented encoding of compound functions generates clauses that contain variables belonging to f_i in combination with variables from preceding functions. This makes it impossible to formulate the encoding as a non-prenex formula where quantifiers such as $\exists v$ from the definition of f_i remain local to the encoding of f_i. A possible solution to this problem will be presented in the following section. In the case of the defined function f_k, quantification is not a problem, since $f_k \approx \Phi$ implies $Q\, f_k \approx Q\, \Phi$.

6. Transformation from QBF to NBF

We now have an efficient transformation from NBF to QBF. Interestingly, a transformation in the other direction is very easy to understand and implement, but seems to require a slight length increase for technical reasons. The idea is to simulate the expansion of quantifiers by suitable function definitions. Let $\exists x\ \phi(x, z_1, ..., z_r)$ be an existentially quantified Boolean formula with a propositional matrix ϕ 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 logically equivalent to the existentially quantified formula. Universal quantifiers can be handled analogously by the dual expansion $\forall x\ \phi(x, z_1, ..., z_r) \approx \phi(0, z_1, ..., z_r) \wedge \phi(1, z_1, ..., z_r)$ and a function term $g_\forall(f(0, z_1, ..., z_r), f(1, z_1, ..., z_r))$ with $g_\forall(a, b) := a \wedge b$.

In propositional logic, we would have to explicitly write down the two expanded terms $\phi(0, z_1, ..., z_r)$ and $\phi(1, z_1, ..., z_r)$ as propositional formulas, which would cause exponential growth when applied multiple times, but with nested Boolean functions, it is sufficient to define $f(x, z_1, ..., z_r) := \phi(x, z_1, ..., z_r)$ only once. We need, however, two copies of the function symbol and its arguments. Since arguments are also counted when determining the length of a NBF, the resulting transformation is not linear, but requires size $O(|v| \cdot |\Phi|)$ for a QBF formula Φ with $|v|$ quantified variables. We call this *v-linear*. It is not clear whether a truly linear transformation from QBF to NBF is also possible.

Proposition 2. For every formula $\Phi \in$ QBF, there exists a nested Boolean function $D(f_k) = (f_0, ..., f_k)$ with $\Phi \approx f_k$, and the length of $D(f_k)$ is v-linear in the length of Φ, that means $O(|v| \cdot |\Phi|)$ if Φ contains $|v|$ quantified variables.

By the simulation of quantifiers, we can also handle the previously suggested extension of compound definitions with quantifiers. For example, a definition of the form

$$f_i(\mathbf{x}^i) := \exists v\ f_{j_0}(f_{j_1}(v, \mathbf{x}^i_1), ..., f_{j_r}(v, \mathbf{x}^i_r))$$

would then be rewritten into two definitions $f_i^{(1)}, f_i^{(2)}$:

$$\begin{array}{lll} f_i^{(1)}(v, \mathbf{x}^i) & := & f_{j_0}(f_{j_1}(v, \mathbf{x}^i_1), ..., f_{j_r}(v, \mathbf{x}^i_r)) \\ f_i^{(2)}(\mathbf{x}^i) & := & g_\exists(f_i^{(1)}(0, \mathbf{x}^i), f_i^{(1)}(1, \mathbf{x}^i)) \end{array}$$

7. NBF Modeling Example

We would like to finish the discussion of NBF with a short example that illustrates the potential of NBF as a modeling language. The problem which we consider is a variant of the well-known bounded reachability or s-t-reachability problem for directed graphs $G = (V, E)$. We consider the nodes as vectors of Boolean variables, e.g. $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$ for a graph with four vertices. The edges are implicitly given by a graph transition relation δ which is represented as a propositional formula. For the four vertices from above, consider the example $\delta(\mathbf{u}, \mathbf{v}) := \delta((u_1, u_2), (v_1, v_2)) := (u_2 \wedge \neg v_1 \wedge \neg v_2) \vee (u_1 \wedge \neg u_2 \wedge v_2)$. There is an edge between two vertices (u_1, u_2) and (v_1, v_2) if and only if $\delta((u_1, u_2), (v_1, v_2)) = 1$. Then our sample graph has four edges, $((0, 1), (0, 0))$, $((1, 1), (0, 0))$, $((1, 0), (0, 1))$ and $((1, 0), (1, 1))$. Given two nodes $\mathbf{s}, \mathbf{t} \in V$ and some bound $k > 0$, the bounded reachability problem is to determine whether there exists a continuous path from $\mathbf{s}$ to $\mathbf{t}$ in G which has length 2^k.

In QBF, we can use the previously mentioned non-copying iterative squaring pattern to encode the problem into a formula of polynomial size (in k and the size of G) that contains only one copy of the potentially large transition relation δ. We obtain

$$\begin{array}{lll}\Phi_0(\mathbf{a},\mathbf{b}) & := & \delta(\mathbf{a},\mathbf{b}) \\ \Phi_i(\mathbf{a},\mathbf{b}) & := & \exists \mathbf{y} \forall \mathbf{u} \forall \mathbf{v}\ [(((\mathbf{u}=\mathbf{a}) \wedge (\mathbf{v}=\mathbf{y})) \vee ((\mathbf{u}=\mathbf{y}) \wedge (\mathbf{v}=\mathbf{b}))) \to \Phi_{i-1}(\mathbf{u},\mathbf{v})]\end{array}$$

for $i = 1, ..., k$. Notice that this is only a formula scheme which indicates the structure of the resulting QBF formula. For $k = 2$, the actual QBF is the following:

$$\begin{array}{lll}\Phi_2(\mathbf{a},\mathbf{b}) & = & \exists \mathbf{y}_2 \forall \mathbf{u}_2 \forall \mathbf{v}_2\ [(((\mathbf{u}_2=\mathbf{a}) \wedge (\mathbf{v}_2=\mathbf{y}_2)) \vee ((\mathbf{u}_2=\mathbf{y}_2) \wedge (\mathbf{v}_2=\mathbf{b}))) \to \\ & & \exists \mathbf{y}_1 \forall \mathbf{u}_1 \forall \mathbf{v}_1\ [(((\mathbf{u}_1=\mathbf{u}_2) \wedge (\mathbf{v}_1=\mathbf{y}_1)) \vee ((\mathbf{u}_1=\mathbf{y}_1) \wedge (\mathbf{v}_1=\mathbf{v}_2))) \to \\ & & \delta(\mathbf{u}_1,\mathbf{v}_1)]]\end{array}$$

With nested Boolean functions (and the ability to use quantifiers as in Section 6), non-copying iterative squaring can be encoded as:

$$\begin{array}{lll}f_\wedge(x_1,x_2) & := & x_1 \wedge x_2 \\ f_0(\mathbf{a},\mathbf{b}) & := & \delta(\mathbf{a},\mathbf{b}) \\ f_1(\mathbf{a},\mathbf{b}) & := & \exists \mathbf{y}\ f_\wedge(f_0(\mathbf{a},\mathbf{y}), f_0(\mathbf{y},\mathbf{b})) \\ f_2(\mathbf{a},\mathbf{b}) & := & \exists \mathbf{y}\ f_\wedge(f_1(\mathbf{a},\mathbf{y}), f_1(\mathbf{y},\mathbf{b})) \\ & \cdots & \end{array}$$

Unlike the formula scheme for QBF, this expression is the actual NBF. We do not need to merge subterms by universally quantified variables as in QBF, since function definitions are never copied in NBF.

At this point, one could certainly argue that the QBF and NBF representations of bounded reachability are essentially the same, with NBF only having an advantage due to counting instantiations of subexpressions as references instead of copies as in QBF. So, let us slightly modify the given problem: is there a continuous path from $\mathbf{s}$ to $\mathbf{t}$ in G which has length 2^k *and which contains at least one vertex* $\mathbf{x}$ *that satisfies some property* p? That means, is there a path $\mathbf{x}_0, ..., \mathbf{x}_{2^k}$ with $\mathbf{s} = \mathbf{x}_0$, $\mathbf{t} = \mathbf{x}_{2^k}$ and $p(\mathbf{x}_i) = 1$ for some $i \in \{0, ..., 2^k\}$? We assume that p is given as a propositional formula.

In NBF, we can add an additional argument *ok* to the functions $f_0, ..., f_k$ which indicates whether the vertex satisfying p has already been found. Then we have an initial function f_0 defined as follows:

$$f_0(\mathbf{a},\mathbf{b},ok) \ := \ \delta(\mathbf{a},\mathbf{b}) \wedge (ok \vee p(\mathbf{a}) \vee p(\mathbf{b}))$$

Notice that the term on the right is just a propositional expression, and thus contains two copies of p. If we wanted to avoid this, we could define another initial function $f_p(\mathbf{x}) := p(\mathbf{x})$ and let f_0 be a compound function. When dividing a path into two halves, if the *ok* flag is already set, we can set the flag for both halves. Otherwise, if *ok* is not set, the crucial vertex must be in one of the halves, and the *ok* flag can be set in the other half. We obtain the following encoding ($i = 1, ..., k$):

$$\begin{array}{lll}f_i(\mathbf{a},\mathbf{b},ok) & := & \exists \mathbf{y}\ [(f_{i-1}(\mathbf{a},\mathbf{y},ok) \wedge f_{i-1}(\mathbf{y},\mathbf{b},1)) \vee \\ & & (f_{i-1}(\mathbf{a},\mathbf{y},1) \wedge f_{i-1}(\mathbf{y},\mathbf{b},ok))]\end{array}$$

Notice that the vertex $\mathbf{y}$ is part of both halves, so if it is the one satisfying p, it will be found eventually. For simplicity of the encoding, we omit checking $\mathbf{y}$ right away.

Notice that strictly following the syntax definition of NBF prevents us from mixing function instantiations and propositional operations, so we are technically not allowed to write the definition of f_i as above. We would require initial functions $f_\wedge(x_1, x_2) := x_1 \wedge x_2$ and $f_\vee(x_1, x_2) := x_1 \vee x_2$, as well as an intermediate function for each f_i $(i = 1, ..., k)$:

$$\begin{array}{lcl} g_i(\mathbf{a}, \mathbf{b}, ok_l, ok_r) & := & f_\wedge(f_{i-1}(\mathbf{a}, \mathbf{y}, ok_l), f_{i-1}(\mathbf{y}, \mathbf{b}, ok_r)) \\ f_i(\mathbf{a}, \mathbf{b}, ok) & := & \exists \mathbf{y}\ f_\vee(g_i(\mathbf{a}, \mathbf{b}, ok, 1), g_i(\mathbf{a}, \mathbf{b}, 1, ok)) \end{array}$$

We assume that real NBF solvers would allow the user to mix function instantiations and propositional operations and automatically rewrite the input into a proper NBF. Alternatively, our NBF to QBF transformation could be adapted to directly accommodate this richer syntax.

While the above NBF encoding is surprisingly short and simple, the same idea is considerably harder to implement in QBF. Non-copying iterative squaring in QBF relies on the idea of reusing variables in both halves of a split in the search space, which makes it difficult to pass different arguments to both subproblems. Our idea is to introduce for each split an existentially quantified variable l which is true when we should look for the crucial vertex in the left half. In addition, we need a universally quantified variable ok to pass the flag to both subproblems. In total, we obtain, the following QBF formula scheme:

$$\begin{array}{lcl} \Phi_0(\mathbf{a}, \mathbf{b}, ok_1) & := & \delta(\mathbf{a}, \mathbf{b}) \wedge (ok_1 \vee p(\mathbf{a}) \vee p(\mathbf{b})) \\ \Phi_i(\mathbf{a}, \mathbf{b}, ok_{i+1}) & := & \exists \mathbf{y}_i \exists l_i \forall \mathbf{u}_i \forall \mathbf{v}_i \forall ok_i\ [\\ & & (((\mathbf{u}_i = \mathbf{a}) \wedge (\mathbf{v}_i = \mathbf{y}_i) \wedge (ok_i = (\neg l_i \vee ok_{i+1})))\vee \\ & & ((\mathbf{u}_i = \mathbf{y}_i) \wedge (\mathbf{v}_i = \mathbf{b}) \wedge (ok_i = (l_i \vee ok_{i+1})))) \\ & & \to \varphi_{i-1}(\mathbf{u}_i, \mathbf{v}_i, ok_i)],\ i = 1, ..., k \\ \Phi(\mathbf{a}, \mathbf{b}) & := & \Phi_k(\mathbf{a}, \mathbf{b}, 0) \end{array}$$

For $k = 2$, the actual QBF is the following:

$$\begin{array}{lcl} \Phi(\mathbf{a}, \mathbf{b}) & = & \exists \mathbf{y}_2 \exists l_2 \forall \mathbf{u}_2 \forall \mathbf{v}_2 \forall ok_2\ [\\ & & (((\mathbf{u}_2 = \mathbf{a}) \wedge (\mathbf{v}_2 = \mathbf{y}_2) \wedge (ok_2 = \neg l_2))\vee \\ & & ((\mathbf{u}_2 = \mathbf{y}_2) \wedge (\mathbf{v}_2 = \mathbf{b}) \wedge (ok_2 = l_2))) \to \\ & & \exists \mathbf{y}_1 \exists l_1 \forall \mathbf{u}_1 \forall \mathbf{v}_1 \forall ok_1\ [\\ & & (((\mathbf{u}_1 = \mathbf{u}_2) \wedge (\mathbf{v}_1 = \mathbf{y}_1) \wedge (ok_1 = \neg l_1 \vee ok_2))\vee \\ & & ((\mathbf{u}_1 = \mathbf{y}_1) \wedge (\mathbf{v}_1 = \mathbf{v}_2) \wedge (ok_1 = l_1 \vee ok_2))) \to \\ & & \delta(\mathbf{u}_1, \mathbf{v}_1) \wedge (ok_1 \vee p(\mathbf{u_1}) \vee p(\mathbf{v_1}))]] \end{array}$$

It should be obvious that this formulation is significantly more complex to understand and write down, and thus more error-prone, than the corresponding NBF formulation. Notice that we are already using the "syntactic sugar" of being able to compare vectors of Boolean variables, which is not needed in the NBF representation.

Things are becoming really complicated with QBF when one subformula occurs with different polarities. For example, we could further modify our reachability problem and add the (admittedly rather artificial) requirement that the vertex in the middle of the path from $\mathbf{s}$ to $\mathbf{t}$, i.e. vertex $\mathbf{x}_{2^{k-1}}$, must not be connected back to $\mathbf{s}$ by a path of length 2^{k-1}. In NBF, this is easy to achieve:

$$\begin{aligned} f_k(\mathbf{a}, \mathbf{b}, ok) \quad := \quad & \exists \mathbf{y}\ [((f_{k-1}(\mathbf{a}, \mathbf{y}, ok) \wedge f_{k-1}(\mathbf{y}, \mathbf{b}, 1)) \vee (f_{k-1}(\mathbf{a}, \mathbf{y}, 1) \wedge f_{k-1}(\mathbf{y}, \mathbf{b}, ok))) \\ & \wedge \neg f_{k-1}(\mathbf{y}, \mathbf{a}, 1)] \end{aligned}$$

In QBF, however, it is difficult to avoid embedding a negated copy of Φ_{k-1} in addition to the positive Φ_{k-1}. We need to apply some of the tricks we used in our NBF to QBF transformation: store the value of Φ_{k-1} in an existential variable and restrict its polarity according to $\mathbf{u}_k, \mathbf{v}_k$ and ok_k. For $k = 2$, the resulting formula is already quite complex:

$$\begin{aligned} \Phi(\mathbf{a}, \mathbf{b}) \quad = \quad & \exists \mathbf{y}_2 \exists l_2 \forall \mathbf{u}_2 \forall \mathbf{v}_2 \forall ok_2 \exists r\ [\\ & (((\mathbf{u}_2 = \mathbf{a}) \wedge (\mathbf{v}_2 = \mathbf{y}_2) \wedge (ok_2 = \neg l_2)) \rightarrow r) \wedge \\ & (((\mathbf{u}_2 = \mathbf{y}_2) \wedge (\mathbf{v}_2 = \mathbf{b}) \wedge (ok_2 = l_2)) \rightarrow r) \wedge \\ & (((\mathbf{u}_2 = \mathbf{y}_2) \wedge (\mathbf{v}_2 = \mathbf{a}) \wedge ok_2) \rightarrow \neg r) \wedge \\ & (r \leftrightarrow (\exists \mathbf{y}_1 \exists l_1 \forall \mathbf{u}_1 \forall \mathbf{v}_1 \forall ok_1\ [\\ & (((\mathbf{u}_1 = \mathbf{u}_2) \wedge (\mathbf{v}_1 = \mathbf{y}_1) \wedge (ok_1 = \neg l_1 \vee ok_2)) \vee \\ & ((\mathbf{u}_1 = \mathbf{y}_1) \wedge (\mathbf{v}_1 = \mathbf{v}_2) \wedge (ok_1 = l_1 \vee ok_2))) \rightarrow \\ & \delta(\mathbf{u}_1, \mathbf{v}_1) \wedge (ok_1 \vee p(\mathbf{u_1}) \vee p(\mathbf{v_1}))]))] \end{aligned}$$

Clearly, such QBF instances should not be constructed manually, and that is the point of our NBF to QBF transformation: specify on a higher level in NBF how subformulas are instantiated and let an automated transformation implement suitable subformula-sharing on the QBF level.

8. Conclusion

We have seen how existing QBF encoding techniques for the compression of propositional formulas can be combined and generalized into an encoding pattern for nested instantiations of propositional subformulas with different arguments. For example, a propositional formula of the form $\alpha(\phi(A_1, B_1), \beta(\phi(A_2, B_2))) \wedge \beta(\phi(A_3, B_3))$ can be represented in a systematic way as a logically equivalent QBF formula which contains only one copy of each subformula α, β and ϕ.

We obtain an equivalence-preserving transformation in linear time from the PSPACE-complete language of nested Boolean functions or Boolean programs to prenex QBF. Considering that the paradigm of function composition is rather different from quantification, it might be more intuitive for some application problems to encode (parts of) the problems as NBF rather than QBF. That means nested Boolean functions could be used as an alternative input language for QBF solvers.

A transformation in the other direction from QBF to NBF is quite easy by the simulation of quantifier expansion, but there are currently no NBF solvers available. Trying to design and implement a solver for NBF seems to be an interesting objective for future work, because the different paradigm of NBF might open new perspectives and lead to the discovery of new solving techniques that could also be valuable for QBF solvers.

References

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

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

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

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

[5] T. Jussila and A. Biere. Compressing BMC encodings with QBF. *Electronic Notes in Theoretical Computer Science*, **174**(3):45–56, 2007.

[6] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. 10th European Conf. on Artificial Intelligence (ECAI 1992)*, pages 359–363. John Wiley and Sons, 1992.

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

[8] H. Kleine Büning, X. Zhao, and U. Bubeck. Transformations into normal forms for quantified circuits. In *Proc. 14th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2011)*, **6695** of *LNCS*, pages 245–258. Springer, 2011.

[9] A. Meyer and L. Stockmeyer. Word problems requiring exponential time, preliminary report. In *Proc. 5th ACM Symp. on Theory of Computing (STOC 1973)*, pages 1–9. ACM, 1973.

[10] C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The seventh QBF solvers evaluation (QBFEVAL 2010). In *Proc. 13th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2010)*, **6175** of *LNCS*, pages 237–250. Springer, 2010.

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

[12] A. Sabharwal, C. Ansotegui, C. Gomes, J. Hart, and B. Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In *Proc. 9th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2006)*, **4121** of *LNCS*, pages 382–395. Springer, 2006.

[13] A. Skelley. Propositional PSPACE reasoning with Boolean programs versus quantified Boolean formulas. In *Proc. 31st Intl. Colloquium on Automata, Languages and Programming (ICALP 2004)*, **3142** of *LNCS*, pages 1163–1175. Springer, 2004.

[14] G. Tseitin. On the complexity of derivation in propositional calculus. In A. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Springer, 1970.