The multiplicative complexity of interval checking

Thomas Häner and Mathias Soeken
Microsoft Quantum, Switzerland

Abstract

We determine the exact AND-gate cost of checking if $a \leq x < b$, where a and b are constant integers. Perhaps surprisingly, we find that the cost of interval checking never exceeds that of a single comparison and, in some cases, it is even lower.

1 Introduction

The multiplicative complexity of a Boolean function f is the smallest number of AND gates in any logic network over the gate set {AND, XOR, NOT} that implements f. Multiplicative complexity is used as an important characteristic metric to measure the cost of cryptographic implementations in secure computation protocols [1, 7, 10] or the cost of fault-tolerant implementations of quantum operations [11]. Unfortunately, computing the multiplicative complexity is intractable [9] and for a random n-variable Boolean function f it is at least $2^{n/2} - O(n)$ with high probability [3]. However, for several families of Boolean functions the multiplicative complexity has been analyzed, including quadratic functions [12], all functions up to 6 variables [5], all functions with a multiplicative complexity of at most 4 [6], all symmetric functions [4], and the Hamming weight function [2].

In this paper, we determine the multiplicative complexity of the interval check $[a \leq x < b]$, where a and b are two nonnegative constant integers and x is an n-bit nonnegative integer. We derive an upper bound on the multiplicative complexity by proposing a construction to implement the interval check, and we derive a matching lower bound based on the algebraic degree of the function (i.e., the largest monomial in its algebraic normal form), thus proving that our construction is optimal with respect to the number of AND gates. We state our main result in Theorem 1.

Theorem 1 (Main result)**.** *Let $n > 0$ be the number of bits, and let a, b be two constant integers $a < b < 2^n$. Let j_a and j_b denote the number of trailing zeros[1] in the binary representation of a and b, respectively. Then, the interval check $a \leq x < b$ for some arbitrary n-bit nonnegative integer x has a multiplicative complexity of*

$$\begin{cases} n - \min\{j_a, j_b\} - 1 & \text{if } j_a \neq j_b, \\ n - j_a - 2 & \text{otherwise.} \end{cases} \tag{1}$$

We present our method for interval checking using an optimal number of AND gates in Section 3. We then analyze our construction and give a proof of Theorem 1 in Section 4.

2 Background

In this section, we introduce algebraic normal forms, algebraic degree, and AND/OR chains, which are a family of Boolean functions central to the implementation of comparison with constants. We use $\#S$ to denote the cardinality of some set S, and we use $\mathrm{ite}(x, f, g) := (x \wedge f) \oplus (\bar{x} \wedge g)$ to denote the *if-then-else* function.

Definition 1 (Algebraic normal form)**.** *Let $S = \{1, \ldots, n\}$ and $x_i \in \{0, 1\}$. Then*

$$f(x_1, \ldots, x_n) = \bigoplus_{I \subseteq S} a_I \bigwedge_{i \in I} x_i \tag{2}$$

is the algebraic normal form *(ANF) of f for some assignment to the coefficients $a_I \in \{0, 1\}$. Each AND-term in (2), where $a_I = 1$, is called a* monomial *of f.*

Example 1. *The ANF of $x_1 \vee x_2$ is $x_1 \oplus x_2 \oplus (x_1 \wedge x_2)$. Here $a_\emptyset = 0$, but $a_{\{1\}} = a_{\{2\}} = a_{\{1,2\}} = 1$.*

[1] The number of trailing zeros in the binary representation of $a \geq 0$ is the largest integer $j \leq n$ such that $\frac{a}{2^j}$ is an integer.

Note that every Boolean function has a unique ANF, since there are 2^n coefficients in (2) and there exist 2^{2^n} Boolean functions over n variables.

Definition 2 (Algebraic degree)**.** *The* algebraic degree *of a Boolean function f is*

$$\deg(f) = \max\{\#I \mid a_I = 1\}, \tag{3}$$

where the coefficients a_I are given by the ANF of f. In other words, the algebraic degree of f is given by the number of variables in its largest monomial.

Definition 3 (Multiplicative complexity)**.** *Let f denote a Boolean function. The multiplicative complexity of f, denoted by $c_\wedge(f)$, is the minimal number of AND gates in any logic network for f over the gate set $\{\wedge, \oplus, \neg\}$, which consists of the 2-input AND and XOR gates and inverters.*

Note that OR gates can be considered AND gates in the context of the multiplicative complexity (see also Example 1).

Lemma 1 (Proposition 3.8, [13])**.** *We have for all Boolean functions f, $c_\wedge(f) \geq \deg(f) - 1$.*

Definition 4 (AND/OR chain)**.** *Given Boolean variables $x_1, x_2, \ldots, x_k$, $k \geq n$, an* AND/OR chain *is any formula*

$$f = x_1 \circ_1 (x_2 \circ_2 (\cdots (x_{n-1} \circ_{n-1} x_n) \cdots)), \tag{4}$$

where $\circ_i \in \{\wedge, \vee\}$. We refer to $\ell(f) = n - 1$ as the length *of f.*

Lemma 2. *Let f be an AND/OR chain. Then $\deg(f) = \ell(f) + 1$ and $c_\wedge(f) = \ell(f)$.*

Proof. We prove that $\deg(f) = \ell(f) + 1$ by induction over the length ℓ of an AND/OR chain. The statement holds trivially for $\ell = 0$. Assuming that the statement holds for ℓ, consider a function $x \circ f$, where $\circ \in \{\wedge, \vee\}$ and f is an AND/OR chain and $\ell(f) = \ell$. Further, x is not in the support of f. Then, we have $\deg(x \circ f) = \deg((x \oplus f)[\circ = \vee] \oplus (x \wedge f)) = \deg(x \wedge f) = 1 + \deg(f) = \ell(f) + 2$. Since the number of operators in f is $\ell(f)$, using Lemma 1, we have $c_\wedge(f) = \ell(f)$. □

3 Construction

In this section, we describe an algorithm to construct a logic network to evaluate $[a \leq x < b]$. A straightforward upper bound on the number of AND gates is the sum of the costs of both individual comparisons. However, we will present a construction that incurs at most the cost of the more costly comparison, and we show that it is possible to save an additional AND gate if both comparisons have identical costs.

3.1 Comparison

As a starting point, we present a construction for comparing an integer to a constant, i.e., evaluating $[a \leq x]$ for a constant integer a and an n-bit nonnegative integer x. We are not aware of any previous work that describes this construction.

Lemma 3. *Let $a = (a_1 \ldots a_{n-1}1)_2$ be an odd constant integer and let $x = (x_1 \ldots x_n)_2$ be an arbitrary n-bit nonnegative integer. Then, the AND/OR chain $x_1 \circ_1 (x_2 \circ_2 (\cdots (x_{n-1} \circ_{n-1} x_n) \cdots))$, with*

$$c_i = \begin{cases} \vee & \text{if } a_i = 0, \\ \wedge & \text{if } a_i = 1, \end{cases}$$

evaluates $[a \leq x]$.

Proof. We prove the statement using induction over n. For $n = 1$, we have $a = 1$ and $x = x_1$, and $[1 \leq x] = x_1$. We assume that the statement holds for any odd constant integer of length n. Consider a constant $a = a_1 2^n + a'$ for some constant integer a' of length n. If $a_1 = 0$, then $[a' \leq x] = x_1 \vee [a' \leq (x_2 \ldots x_{n+1})_2]$. If $a_1 = 1$, then $[2^n + a' \leq x] = x_1 \wedge [a' \leq (x_2 \ldots x_{n+1})_2]$. □

We provide pseudocode for this construction in Listing 1, where we use the $\leftarrow$-operator to denote insertion of the expression on the right into the formula on the left at the unique position identified by the $\cdot$ symbol. Invoking `comparison_formula(a, range={1,n})` produces the AND/OR chain from Lemma 3 for evaluating $[a \leq x]$, where x is an n-bit integer and a is an odd n-bit constant integer.

Note that comparing to an even number can be recast as a comparison against an odd number. This allows us to derive the multiplicative complexity of comparison using Lemma 2, leading to the following corollary.

Corollary 1. *Let $a = 2^j k > 0$ be a constant integer, where k is odd. Evaluating $[a \leq x]$ is equivalent to evaluating $[a/2^j \leq (x_1 \ldots x_{n-j})_2]$, and therefore $c_\wedge([a \leq x]) = n - j - 1$.*

```
Formula comparison_formula(a, range):
    F ← (·) // formula being constructed
    for (k = range.low; k < range.high; ++k)
        ∘ = a[k] ? ∧ : ∨ // operator is chosen according to a[k]
        F ← x_k ∘ (·)
    return (F ← x_range.high)
```

Listing 1: Pseudocode for generating a formula for comparison $[a \leq x]$ (assuming a is odd) using our construction. The `range` argument can be used to generate subchains, which will be useful for our interval check construction. In the code, `a[k]` denotes the k-th bit of the n-bit integer a, with k ranging from 1 to n and `a[1]` being the most-significant bit.

3.2 Interval checking

Next, we construct a logic network for interval checking. To this end, note that $[a \leq x < b]$ is equivalent to

$$[a \leq x] \oplus [b \leq x], \tag{5}$$

since $a < b$.[2] Starting from this expression, our algorithm for constructing the interval check proceeds by iteratively decomposing $[a \leq x] \oplus [b \leq x] = (x_1 \circ f_1) \oplus (x_1 \bullet f_2)$, where both f_1 and f_2 are either AND/OR chains or constants, and $\circ$ and $\bullet$ are either $\wedge$ or $\vee$. The variables involved in f_1 and f_2 are either the same and appear in the same order, or the variables in one chain are a prefix of the variables in the other chain. We show how to evaluate this expression for different choices of $\circ$ and $\bullet$, and that this leads to a formula involving either $f_1 \oplus f_2$ (allowing us to recurse), or the if-then-else (ite) operation on two AND/OR chains.

Specifically, if f_1 or f_2 is a constant, then

$$x \oplus (x \wedge f) = x \wedge \bar{f} \quad \text{or} \quad x \oplus (x \vee f) = \bar{x} \wedge f, \tag{6}$$

and the iteration stops. Otherwise, one of the following three cases applies:

$$(x \wedge f_1) \oplus (x \wedge f_2) = x \wedge (f_1 \oplus f_2) \tag{7a}$$
$$(x \vee f_1) \oplus (x \vee f_2) = \bar{x} \wedge (f_1 \oplus f_2) \tag{7b}$$
$$(x \vee f_1) \oplus (x \wedge f_2) = \mathrm{ite}(x, \bar{f}_2, f_1) \tag{7c}$$

Proof. Equations (6) and (7a) follow from straightforward Boolean identities. For (7b), note that $(x \vee f_1) \oplus (x \vee f_2) = (\bar{x} \wedge \bar{f}_1) \oplus (\bar{x} \wedge \bar{f}_2) = \bar{x} \wedge (f_1 \oplus f_2)$, by applying De Morgan's law and using the fact that $\bar{x} \oplus \bar{x} = x \oplus x$. By expanding the first term in (7c) into an ANF, one obtains $f_1 \oplus (x \wedge f_1) \oplus x \oplus (x \wedge f_2)$. Then the first two and the last two terms can be merged into $(\bar{x} \wedge f_1) \oplus (x \wedge \bar{f}_2) = \mathrm{ite}(x, \bar{f}_2, f_1)$. □

Next, we show a special construction for $\mathrm{ite}(x, \bar{f}_2, f_1)$ that exploits the fact that f_1 and f_2 are both AND/OR chains. Their formulas only differ in their lengths or in what operators are used. The idea is to propagate the inversion of f_2 into the formula (using De Morgan's laws) in order to make the operators match those of f_1. Then, the inversions and potential chain suffixes (for chains with unequal lengths) can be implemented in the same formula conditional on x.

Before proving the general case, we discuss the following examples to provide some intuition. In the first example, both chains have the same lengths and none of the operators are equal.

Example 2. *Let* $f_1 = x_2 \vee (x_3 \wedge x_4)$ *and* $f_2 = x_2 \wedge (x_3 \vee x_4)$. *Then*

$$\begin{aligned} \bar{f}_2 &= \overline{x_2 \wedge (x_3 \vee x_4)} \\ &= \bar{x}_2 \vee \overline{(x_3 \vee x_4)} \\ &= \bar{x}_2 \vee (\bar{x}_3 \wedge \bar{x}_4), \end{aligned}$$

and therefore

$$\mathrm{ite}(x_1, \bar{f}_2, f_1) = (x_1 \oplus x_2) \vee ((x_1 \oplus x_3) \wedge (x_1 \oplus x_4)).$$

Note how this formula evaluates to f_1, *if* $x_1 = 0$, *and to* $\bar{f}_2$, *if* $x_1 = 1$.

In the second example, the chain lengths are still equal, but one operator is the same: The $\vee$ operator links x_2 to the rest of the chain in both f_1 and f_2.

[2] This expression could also be evaluated on a quantum computer using Deutsch's algorithm [8] with a single comparator.

Example 3. *Let* $f_1 = x_2 \vee (x_3 \wedge x_4)$ *and* $f_2 = x_2 \vee (x_3 \vee x_4)$*. Then*

$$\begin{aligned}\bar{f}_2 &= \overline{x_2 \vee (x_3 \vee x_4)} \\ &= \overline{x_2 \vee \overline{\overline{(x_3 \vee x_4)}}} \\ &= \overline{x_2 \vee \overline{(\bar{x}_3 \wedge \bar{x}_4)}},\end{aligned}$$

and therefore

$$\mathrm{ite}(x_1, \bar{f}_2, f_1) = x_1 \oplus (x_2 \vee (x_1 \oplus ((x_1 \oplus x_3) \wedge (x_1 \oplus x_4)))).$$

Note how the inverter is not propagated when the operators are the same, but a double negation is introduced to further propagate the inverter to the remaining part of the chain.

In the final example, we consider the case in which one chain is longer than the other.

Example 4. *Let* $f_1 = x_2 \wedge (x_3 \vee x_4)$ *and* $f_2 = x_2 \vee (x_3 \wedge (x_4 \vee x_5))$*. Then*

$$\begin{aligned}\bar{f}_2 &= \overline{x_2 \vee (x_3 \wedge (x_4 \vee x_5))} \\ &= \bar{x}_2 \wedge \overline{(x_3 \wedge (x_4 \vee x_5))} \\ &= \bar{x}_2 \wedge (\bar{x}_3 \vee \overline{(x_4 \vee x_5)}),\end{aligned}$$

and therefore

$$\mathrm{ite}(x_1, \bar{f}_2, f_1) = (x_1 \oplus x_2) \wedge ((x_1 \oplus x_3) \vee (x_1 \oplus (x_4 \vee (x_1 \wedge x_5)))).$$

Note how x_1 *is not only used to invert subterms of the formula, but also to conditionally include* x_5 *to represent* $\bar{f}_2$*.*

We can now enumerate all cases for f_1 and f_2 in $\mathrm{ite}(x, \bar{f}_2, f_1)$. In the following, $x_i \neq x_j$, and neither of the two variables occurs in f_1 or f_2. The terminal cases apply when $\ell(f_1) = 0$ or $\ell(f_2) = 0$:

$$\mathrm{ite}(x_i, \bar{x}_j, x_j) = x_i \oplus x_j \tag{8a}$$
$$\mathrm{ite}(x_i, \bar{x}_j, x_j \wedge f) = (x_i \oplus x_j) \wedge (x_i \vee f) \tag{8b}$$
$$\mathrm{ite}(x_i, \bar{x}_j, x_j \vee f) = (x_i \oplus x_j) \vee (\bar{x}_i \wedge f) \tag{8c}$$
$$\mathrm{ite}(x_i, \overline{x_j \wedge f}, x_j) = (x_i \oplus x_j) \vee (x_i \wedge \bar{f}) \tag{8d}$$
$$\mathrm{ite}(x_i, \overline{x_j \vee f}, x_j) = (x_i \oplus x_j) \wedge (\bar{x}_i \vee \bar{f}) \tag{8e}$$

In addition to these 5 terminal cases, there are 4 non-terminal cases:

$$\mathrm{ite}(x_i, \overline{x_j \vee f_2}, x_j \wedge f_1) = (x_i \oplus x_j) \wedge \mathrm{ite}(x_i, \bar{f}_2, f_1) \tag{9a}$$
$$\mathrm{ite}(x_i, \overline{x_j \wedge f_2}, x_j \vee f_1) = (x_i \oplus x_j) \vee \mathrm{ite}(x_i, \bar{f}_2, f_1) \tag{9b}$$
$$\mathrm{ite}(x_i, \overline{x_j \wedge f_2}, x_j \wedge f_1) = x_i \oplus (x_j \wedge (x_i \oplus \mathrm{ite}(x_i, \bar{f}_2, f_1))) \tag{9c}$$
$$\mathrm{ite}(x_i, \overline{x_j \vee f_2}, x_j \vee f_1) = x_i \oplus (x_j \vee (x_i \oplus \mathrm{ite}(x_i, \bar{f}_2, f_1)) \tag{9d}$$

These identities are readily verified by expressing both sides of the equation as an ANF and using identities such as $x \oplus x = 0$ and $\bar{x} \wedge y = y \oplus (x \wedge y)$.

We provide pseudocode for our construction in Listing 2, where we make use of the shorthand x^b for $x, b \in \{0, 1\}$ to mean

$$x^b = \begin{cases} x & \text{if } b = 1, \\ \overline{x} & \text{if } b = 0, \end{cases}$$

and for a binary operator $\circ \in \{\wedge, \vee\}$, we denote by $\overline{\circ}$ its dual operator, i.e., if $\circ = \wedge$, then $\overline{\circ} = \vee$ and vice-versa.

```
Formula interval_check_formula(n, a, b, j_a, j_b):
    cutoff = n - min(j_a, j_b)
    F = (·) // formula being constructed
    // first, remove identical prefix: Eqs. (7a), (7b)
    for (i = 1; a[i] == b[i] and i < n - max(j_a, j_b); ++i)
        F ← x_i^{a[i]} ∧ (·)

    // iteration stops if f_1 or f_2 is a constant: Eq. (6)
    if i == n - j_a
        f = comparison_formula(b, range={i+1,cutoff})
        return (F ← x_i ∧ f̄)
    if i == n - j_b
        f = comparison_formula(a, range={i+1,cutoff})
        return (F ← x̄_i ∧ f)

    // handle remaining and/or-chain sections with ite()-recursion: Eqs. (9a) – (9d)
    for (j = i+1; j < n - max(j_a, j_b); ++j)
        ∘_j = a[j] ? ∧ : ∨ // and/or is chosen according to a[j]
        if a[j] == b[j] // same operators in and/or chains
            F ← x_i ⊕ (x_j ∘_j (x_i ⊕ (·)))
        else // different operators in and/or chains
            F ← (x_i ⊕ x_j) ∘_j (·)

    // handle terminal cases for ite(): Eqs. (8a) – (8e)
    if j_a != j_b // unequal lengths
        negop = j_a > j_b ? !b[j] : a[j]
        ∘ = negop ? ∧ : ∨
        num = j_a > j_b ? b : a
        // get the postfix of the longer chain
        f = comparison_formula(num, range={j+1,cutoff})
        // and merge with the formula
        if j_a > j_b
            negop = !negop
        return (F ← (x_i ⊕ x_j) ∘ x_i^{negop} ∘̄ f^{j_a<j_b})
    else // Eq. (8a)
        return (F ← x_i ⊕ x_j)
```

Listing 2: Pseudocode for generating a formula for the interval check $[a \leq x < b]$ based on the construction detailed in Section 3. As in Listing 1, `a[k]` denotes the k-th bit of the n-bit integer a and `a[1]` is the most-significant bit. `j_a` and `j_b` correspond to j_a and j_b, respectively.

4 Analysis

In this section, we determine the multiplicative complexity of the interval check $[a \leq x < b]$. To this end, we first compute an upper bound on the multiplicative complexity by counting the number of AND and OR gates that appear in our construction from the previous section. Then, we determine a lower bound on the multiplicative complexity by deriving the algebraic degree of our construction. We will conclude that the lower bound matches the upper bound, allowing us to prove Theorem 1.

4.1 Upper bound

In the following, let $u(f)$ be the number of AND/OR gates that are applied when evaluating f according to the construction discussed in the previous section.

Lemma 4. *Let f_1 and f_2 be two AND/OR chains that do not contain x. Then*

$$u(\mathrm{ite}(x, \bar{f}_2, f_1)) = \begin{cases} \ell(f_1) & \text{if } \ell(f_1) = \ell(f_2), \\ \max\{\ell(f_1), \ell(f_2)\} + 1 & \text{otherwise.} \end{cases}$$

Proof. W.l.o.g. we assume that $\ell(f_2) \geq \ell(f_1)$ and prove the statement by induction on $\ell(f_1)$. In the base case, $\ell(f_1) = 0$, the statement follows from (8a) when $\ell(f_2) = 0$. Otherwise, either (8d) or (8e) applies, resulting in $2 + \ell(f) = 2 + \ell(f_2) - 1 = \ell(f_2) + 1$ AND or OR gates. For the induction step, assume that the statement holds for $\ell(f_1) = \ell$ and let f_1' be an AND/OR chain with $\ell(f_1') = \ell + 1$. Then, one of the cases in (9a)–(9d) must apply, which adds one AND or OR gate in each case and reduces the length of the formulas passed to 'ite' by one. □

Lemma 5. *Let f_1 and f_2 be two distinct AND/OR chains. Then,*

$$u(f_1 \oplus f_2) = \begin{cases} \ell(f_1) - 1 & \text{if } \ell(f_1) = \ell(f_2), \\ \max\{\ell(f_1), \ell(f_2)\} & \text{otherwise.} \end{cases}$$

Proof. We assume w.l.o.g. that $\ell(f_2) \geq \ell(f_1)$. If the first k operators (indexed from 1 to k in (4)) in the AND/OR chains of f_1 and f_2 are the same, we can apply equations (7a) and (7b) k times, resulting in $u(f_1 \oplus f_2) = k + u(f_1' \oplus f_2')$, where f_1' and f_2' are obtained by removing the first k operators of f_1 and f_2, respectively, with $\ell(f_1') = \ell(f_1) - k$, and $\ell(f_2') = \ell(f_2) - k$. Note that $f_1' \neq f_2'$, since $f_1 \neq f_2$. Therefore, it is sufficient to consider the case in which the top most operators of f_1' and f_2' differ or the case in which $\ell(f_1') = 0$. In the first case, $f_1' \oplus f_2'$ can be written as an if-then-else construct acting on chains that are shortened by 1 operand/operator, see (7c), and the statement follows from Lemma 4. In the second case, the statement follows from (6). □

4.2 Lower bound

We prove a lower bound on the multiplicative complexity by computing the algebraic degree of $f_1 \oplus f_2$, where f_1 and f_2 are AND/OR chains. To do so, we first prove the following fact for the case where $\ell(f_1) = \ell(f_2)$.

Lemma 6. *Let f_1 and f_2 be two distinct AND/OR chains of identical length $n-1$. Then, the largest monomial of f_1 is of length n and equal to that of f_2, and there exists a monomial of length $n-1$ that exists in only one of the two chains.*

Proof. Let f denote an AND/OR chain with inputs $x_2, ..., x_n$. Noting that $x_1 \vee f = x_1 \oplus f \oplus (x_1 \wedge f)$, the largest monomial of f_1 and f_2 is $(x_1 \wedge x_2 \wedge \cdots \wedge x_n)$. To prove the second statement, note that since $f_1 \neq f_2$, there exists a largest position $k \leq n-1$ at which the operators in f_1 and f_2 differ. W.l.o.g., f_1 has an $\vee$-operator and f_2 has an $\wedge$-operator in the k-th position. We write $f_1 = f_1' \circ (x_k \vee g(x_{k+1}, \ldots, x_n))$ and $f_2 = f_2' \bullet (x_k \wedge g(x_{k+1}, \ldots, x_n))$, where f_1' and f_2' are prefixes involving variables $x_1, \ldots, x_{k-1}$ and g is an AND/OR chain with the largest monomial $(x_{k+1} \wedge \cdots \wedge x_n)$. In turn, this is one of the second largest monomials in the ANF of $x_k \vee g = x_k \oplus g \oplus x_k \wedge g$, but not in the ANF of $x_k \wedge g$, where all monomials feature the variable x_k. Consequently, note that f_1 features the monomial $x_1 \wedge \cdots \wedge x_{k-1} \wedge x_{k+1} \wedge \cdots \wedge x_n$, which is not present in the ANF of f_2. □

Using Lemma 6, we can compute the algebraic degree of $f_1 \oplus f_2$ as follows.

Lemma 7. *Let f_1 and f_2 be two distinct AND/OR chains. Then,*

$$\deg(f_1 \oplus f_2) = \begin{cases} \ell(f_1) & \text{if } \ell(f_1) = \ell(f_2), \\ \max\{\ell(f_1), \ell(f_2)\} + 1 & \text{otherwise.} \end{cases}$$

Proof. In the case $\ell(f_1) \neq \ell(f_2)$, assume w.l.o.g. that $\ell(f_1) > \ell(f_2)$. The degree of f_1 is $\ell(f_1) + 1$ (see Lemma 2) since its single largest monomial contains all variables in the support of f_1. All monomials of f_2 are smaller, and therefore $\deg(f_1 \oplus f_2) = \deg(f_1) = \ell(f_1) + 1$.

If $\ell(f_1) = \ell(f_2)$, then Lemma 6 implies that the largest monomials of f_1 and f_2 are identical and thus no longer present in the ANF of $f_1 \oplus f_2$, and that there exists a monomial of length $\ell(f_1)$ that is present in only one of the two ANFs and thus also in the ANF of $f_1 \oplus f_2$. Therefore, $\deg(f_1 \oplus f_2) = \deg(f_1) - 1 = l(f_1)$. □

4.3 Proof of Theorem 1

We are now in a position to prove the main theorem.

Proof. We have

$$\begin{aligned} f = [a \leq x < b] &= [a \leq x] \oplus [b \leq x] \\ &= \underbrace{[(a/2^{j_a}) \leq (x_1 \ldots x_{n-j_a})_2]}_{f_a} \oplus \underbrace{[(b/2^{j_b}) \leq (x_1 \ldots x_{n-j_b})_2]}_{f_b}, \\ &= f_a \oplus f_b \end{aligned}$$

where f_a and f_b are AND/OR chains with a respective length of $n-j_a-1$ and $n-j_b-1$, see Corollary 1. Therefore, $\max\{\ell(f_a), \ell(f_b)\} = n - \min\{j_a, j_b\} - 1$. From Lemma 5, it follows that $c_\wedge(f) \leq n - \min\{j_a, j_b\} - 1 - \delta_{j_a j_b}$, and from Lemma 7 together with Lemma 1, it follows that $c_\wedge(f) \geq \deg(f) - 1 = n - \min\{j_a, j_b\} - 1 - \delta_{j_a j_b}$. Therefore,

$$c_\wedge(f) = n - \min\{j_a, j_b\} - 1 - \delta_{j_a j_b},$$

where δ_{ij} denotes the Kronecker delta. □

5 Conclusions

We have derived the multiplicative complexity of interval checking given two constant bounds. Our construction is of practical interest as it reduces the cost of interval checking by up to a factor of 2 compared to a construction composed of two comparators. This motivates us to study the multiplicative complexity of similar composite operations, e.g., $[x = y \pm a]$ or $[a \leq x + y]$, where a is a constant. This may in turn provide some insight into the multiplicative complexity of other practically-relevant operations such as multiplication, which can be considered a composition of simpler arithmetic operations.

We are further interested in studying the multiplicative complexity of formulas $f = \text{ite}(x, f_1, f_2)$. In general $c_\wedge(f) \leq 1 + c_\wedge(f_1) + c_\wedge(f_2)$, but in this paper we found examples in which the multiplicative complexity of f did not exceed that of the two subformulas. Ideally, we would like to find characteristic properties for f_1 and f_2 that hold if and only if $c_\wedge(f) \leq \max\{c_\wedge(f_1), c_\wedge(f_2)\}$.

References

[1] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In *Int'l Conf. on the Theory and Applications of Cryptographic Techniques*, pages 430–454, 2015.

[2] J. Boyar and R. Peralta. The exact multiplicative complexity of the Hamming weight function. *Electronic Colloquium on Computational Complexity*, (049), 2005.

[3] J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$. *Theoretical Computer Science*, 235(1):43–57, 2000.

[4] L. T. A. N. Brandão, Ç. Çalik, M. S. Turan, and R. Peralta. Upper bounds on the multiplicative complexity of symmetric Boolean functions. *Cryptography and Communications*, 11(6):1339–1362, 2019.

[5] Ç. Çalik, M. S. Turan, and R. Peralta. The multiplicative complexity of 6-variable Boolean functions. *Cryptography and Communications*, 11(1):93–107, 2019.

[6] Ç. Çalik, M. S. Turan, and R. Peralta. Boolean functions with multiplicative complexity 3 and 4. *Cryptography and Communications*, 12(5):935–946, 2020.

[7] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM SIGSAC Conf. on Computer and Communications Security*, pages 1825–1842, 2017.

[8] D. Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proc. R. Soc. Lond.*, A 400(1818):97–117, 1985.

[9] M. G. Find. On the complexity of computing two nonlinearity measures. In *Int'l Computer Science Symposium in Russia*, pages 167–175, 2014.

[10] I. Giacomelli, J. Madsen, and C. Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In *USENIX Security Symposium*, pages 1069–1083, 2016.

[11] G. Meuli, M. Soeken, E. Campbell, M. Roetteler, and G. De Micheli. The role of multiplicative complexity in compiling low T-count oracle circuits. In *Int'l Conf. on Computer-Aided Design*, pages 1–8, 2019.

[12] R. Mirwald and C. Schnorr. The multiplicative complexity of quadratic Boolean forms. In *Foundations of Computer Science*, pages 141–150. IEEE Computer Society, 1987.

[13] C.-P. Schnorr. The multiplicative complexity of Boolean functions. In *Int'l Conf. on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 45–58, 1988.