

Complexity Analysis of Tree Share Structure

Xuan-Bach Le[†] Aquinas Hobor^{‡,*} Anthony W. Lin[†]

[†]University of Oxford [‡]Yale-NUS College *National University of Singapore

Abstract. The tree share structure proposed by Dockins et al. is an elegant model for tracking disjoint ownership in concurrent separation logic, but decision procedures for tree shares are hard to implement due to a lack of a systematic theoretical study. We show that the first-order theory of the full Boolean algebra of tree shares (that is, with all tree-share constants) is decidable and has the same complexity as of the first-order theory of Countable Atomless Boolean Algebras. We prove that combining this additive structure with a constant-restricted unary multiplicative “relativization” operator has a non-elementary lower bound. We examine the consequences of this lower bound and prove that it comes from the combination of both theories by proving an upper bound on a generalization of the restricted multiplicative theory in isolation.

1 Introduction

One general challenge in concurrent program verification is how to specify the ownership of shared resources among threads. A common solution is to tag shared resources with *fractional shares* that track “how much” of a resource is owned by an actor. A *policy* maps ownership quanta with permitted behaviour. For example, a memory cell can be “fully owned” by a thread, permitting both reading and writing; “partially owned”, permitting only reading; or “unowned”, permitting nothing; the initial model of fractional shares [8] was rationals in $[0, 1]$. Since their introduction, many program logics have used a variety of flavors of fractional permissions to verify programs [8, 7, 33, 18, 3, 37, 2, 24, 38, 15, 26, 14].

Rationals do not mix cleanly with concurrent separation logic [31] because they do not preserve the “disjointness” property of separation logic [32]. Dockins *et al.* [13] proposed a “tree share” model that do preserve this property, and so a number of program logics have incorporated them [19, 18, 37, 2, 26].

In addition to their good metatheoretic properties, tree shares have desirable computational properties, which has enabled several highly-automated verification tools to incorporate them [37, 20] via heuristics and decision procedures [25, 28]. As we shall explain in §2.2, tree shares have both “additive” and “multiplicative” substructures. All of the verification tools used only a restricted fragment of the additive substructure (in particular, with only one quantifier alternation) because the general theory’s computational structure was not well-understood. These structures are worthy of further study both because even short programs can require hundreds of tree share entailment queries in the permitted formalism [16, Ch4:§2,§6.4,§6.6], and because recent program logics have shown how the multiplicative structures aid program verification [2, 26].

Recently, Le *et al.* did a more systematic analysis of the computational complexity of certain classes of tree share formulae [27]; briefly:

- the additive structure forms a Countable Atomless Boolean Algebra, giving a well-understood complexity for all first-order formulae *so long as they only use the distinguished constants* “empty” $\mathbf{0}$ and “full” $\mathbf{1}$;
- the multiplicative structure has a decidable existential theory but an undecidable first-order theory; and
- the additive theory in conjunction with a weakened version of the multiplicative theory—in particular, only permitting multiplication by constants on the right-hand side—regained first-order decidability.

Contributions. We address significant gaps in our theoretical understanding of tree shares that deter their use in automated tools for more sophisticated tasks.

- §3 Moving from a restricted fragment of a first-order additive theory to the more general setting of unrestricted first-order formulae over Boolean operations is intuitively appealing due to the increased expressibility of the logic. This expressibility even has computational consequences, as we demonstrate by using it to remove a common source of quantifier alternations. However, verifications in practice often require formulae that incorporate more general constants than $\mathbf{0}$ and $\mathbf{1}$, limiting the application of the analysis from [27] in practice. This is unsurprising since it is true in other settings: many Presburger formulae that arise in engineering contexts, for example, are littered with application-specific constants, *e.g.*, $\forall x.(\exists y.x + y = 7) \Rightarrow (x + 13 < 21)$. A recent benchmark using tree shares for program verification [28] supports this intuition: it made 16k calls in the supported first-order additive fragment, and 21.1% (71k/335k) of the constants used in practice were neither $\mathbf{0}$ nor $\mathbf{1}$. Our main contribution on the additive side is to give a polynomial-time algorithm that reduces first-order additive formulae with arbitrary tree-share constants to first-order formulae using only $\mathbf{0}$ and $\mathbf{1}$, demonstrating that the additive structure’s exact complexity is $\text{STA}(*, 2^{n^{O(1)}}, n)$ -complete and closing the theory/practice gap between [27] and [28].
- §4 We examine the combined additive/restricted multiplicative theory proved decidable in [27]. We prove a nonelementary lower bound for this theory, via a reduction from the combined theory into the string structure with suffix successors and a prefix relation, closing the complexity gap in the theory.
- §5 We investigate the reasons for, and mitigants to, the above nonelementary lower bound. First, we show that the first-order restricted-multiplicative theory on its own (*i.e.*, without the Boolean operators) has elementary complexity via an efficient isomorphism with strings equipped with prefix and suffix successors. Thus, the nonelementary behavior comes precisely from the combination of both theories. Lastly, we examine the kinds of formulae that we expect in practice—for example, those coming from biabduction problems discussed in [26]—and notice that they have elementary complexity.

The other sections of our paper support our contributions by (§2) overviewing tree shares, related work, and several basic complexity results; and by (§6) discussing directions for future work and concluding.

2 Preliminaries

Here we document the preliminaries for our result. Some are standard (§2.1) while others are specific to the domain of tree shares (§2.2–§2.4).

2.1 Complexity preliminaries

We assume that the readers are familiar with basic concepts in computational complexity such as Turing machine, many-one reduction, space and time complexity classes such as NP and PSPACE. A problem is *nonelementary* if it cannot be solved by any deterministic Turing machine that can be time-bounded by one of the exponent functions $\exp(1) = 2^n$, $\exp(n+1) = 2^{\exp(n)}$. Let A, R be complexity classes, a problem P is \leq_R -complete for A iff P is in A and every problem in A is many-one reduced into P via Turing machines in R. In addition, we use $\leq_{R\text{-lin}}$ to assert *linear reduction* that belongs to R and only uses linear space with respect to the problem's size. In particular, $\leq_{\log\text{-lin}}$ is linear log-space reduction. Furthermore, we denote $\text{STA}(p(n), t(n), a(n))$ the class of alternating Turing machine [9] that uses at most $p(n)$ space, $t(n)$ time and $a(n)$ alternations between universal states and existential states or vice versa for input of length n . If any of the three bounds is not specified, we replace it with the symbol $*$, e.g. $\text{STA}(*, 2^{n^{O(1)}}, n)$ is the class of alternating Turing machines that have exponential time complexity and use at most n alternations.

2.2 Overview of tree share structure

A tree share is a binary tree with Boolean leaves \circ (white leaf) and \bullet (black leaf). Full ownership is represented by \bullet and no ownership by \circ . For fractional ownership, one can use, e.g. $\widehat{\bullet\circ}$, to represent the left half-owned resource. Importantly and usefully, $\widehat{\circ\bullet}$ is a distinct tree share representing the other right half. We require tree shares are in canonical form, that is, any subtree $\widehat{\tau\tau}$ where $\tau \in \{\bullet, \circ\}$ needs to be rewritten into τ . For example, both $\widehat{\bullet\circ}$ and $\widehat{\bullet\circ\circ\bullet}$ represent the same tree share but only the former tree is canonical and thus valid. As a result, the set of tree shares \mathbb{T} is a strict subset of the set of all Boolean binary trees. Tree shares are equipped with Boolean operators \sqcup (union), \sqcap (intersection) and $\bar{}$ (complement). When applied to tree shares of height zero, i.e. $\{\bullet, \circ\}$, these operators give the same results as in the case of binary BA. Otherwise, our tree shares need to be unfolded and folded accordingly before and after applying the operators leaf-wise, e.g.

$$\overline{\widehat{\bullet\circ}} = \widehat{\circ\bullet} \quad \widehat{\bullet\circ\bullet} \sqcup \widehat{\circ\circ\bullet} \cong \widehat{\bullet\circ\bullet\bullet} \sqcup \widehat{\circ\circ\circ\bullet} = \widehat{\bullet\circ\bullet\bullet} \cong \widehat{\bullet\circ\bullet}.$$

The additive operator \oplus can be defined using \sqcup and \sqcap , *i.e.* disjoint union:

$$a \oplus b = c \stackrel{\text{def}}{=} a \sqcup b = c \wedge a \sqcap b = \circ.$$

Tree shares also have a multiplicative operator \bowtie called “bowtie”, where $\tau_1 \bowtie \tau_2$ is defined by replacing each black leaf \bullet of τ_1 with an instance of τ_2 , *e.g.*

$$\begin{array}{c} \wedge \\ \bullet \circ \bullet \end{array} \bowtie \begin{array}{c} \wedge \\ \circ \bullet \end{array} = \begin{array}{c} \wedge \\ \circ \bullet \circ \circ \bullet \end{array}.$$

While the \oplus operator has standard additive properties such as commutativity, associativity and cancellativity, the \bowtie operator enjoys the unit \bullet , is associative, injective over non- \circ arguments, and distributes over $\{\sqcup, \sqcap, \oplus\}$ on the left [13]. However, \bowtie is not commutative, *e.g.*:

$$\begin{array}{c} \wedge \\ \bullet \circ \end{array} \bowtie \begin{array}{c} \wedge \\ \circ \bullet \end{array} = \begin{array}{c} \wedge \\ \circ \bullet \circ \end{array} \neq \begin{array}{c} \wedge \\ \circ \bullet \circ \end{array} = \begin{array}{c} \wedge \\ \circ \bullet \end{array} \bowtie \begin{array}{c} \wedge \\ \bullet \circ \end{array}$$

The formalism of these binary operators can all be found in [13].

2.3 Tree shares in program verification

Fractional permissions in general, or tree shares in particular, are integrated into separation logic to reason about ownership. In detail, the mapsto predicate $x \mapsto v$ is enhanced with the permission π , denoted as $x \xrightarrow{\pi} v$, to assert that π is assigned to the address x associated with the value v . This notation of fractional mapsto predicate allows us to split and combine permissions conveniently using the additive operator \oplus and disjoint conjunction \star :

$$x \xrightarrow{\pi_1 \oplus \pi_2} v \dashv\vdash x \xrightarrow{\pi_1} v \star x \xrightarrow{\pi_2} v. \quad (1)$$

The key difference between tree share model $\langle \mathbb{T}, \oplus \rangle$ and rational model $\langle \mathbb{Q}, + \rangle$ is that the latter fails to preserve the disjointness property of separation logic. For instance, while the predicate $x \mapsto 1 \star x \mapsto 1$ is unsatisfiable, its rational version $x \xrightarrow{0.5} 1 \star x \xrightarrow{0.5} 1$, which is equivalent to $x \xrightarrow{1} 1$ by (1), is satisfiable.

On the other hand, the tree share version $x \xrightarrow{\begin{array}{c} \wedge \\ \bullet \circ \end{array}} \star x \xrightarrow{\begin{array}{c} \wedge \\ \bullet \circ \end{array}}$ remains unsatisfiable as the sum $\begin{array}{c} \wedge \\ \bullet \circ \end{array} \oplus \begin{array}{c} \wedge \\ \bullet \circ \end{array}$ is undefined. Such defect of the rational model gives rise to the deformation of recursive structures or elevates the difficulties of modular reasoning, as first pointed out by [32].

Recently, Le and Hobor [26] proposed a proof system for disjoint permissions using the structure $\langle \mathbb{T}, \oplus, \bowtie \rangle$. Their system introduces the notion of predicate multiplication where $\pi \cdot P$ asserts that the permission π is associated with the predicate P . To split the permission, one can apply the following bi-entailment:

$$\pi \cdot P \dashv\vdash (\pi \bowtie \begin{array}{c} \wedge \\ \bullet \circ \end{array}) \cdot P \star (\pi \bowtie \begin{array}{c} \wedge \\ \circ \bullet \end{array}) \cdot P.$$

which requires the following property of tree shares to hold:

$$\forall \pi. \pi = (\pi \bowtie \begin{array}{c} \wedge \\ \bullet \circ \end{array}) \oplus (\pi \bowtie \begin{array}{c} \wedge \\ \circ \bullet \end{array}). \quad (2)$$

Note that the above property demands a combined reasoning of both \oplus and \bowtie . While such property can be manually proved in theorem provers such as Coq [12] using inductive argument, it cannot be handled automatically by known tree share solvers [25, 28] due to the shortness of theoretical insights.

2.4 Previous results on the computational behavior of tree shares

The first sophisticated analysis of the computational properties of tree shares were done by Le *et al.* [27]. They showed that the structure $\langle \mathbb{T}, \sqcup, \sqcap, \bar{\cdot} \rangle$ is a Countable Atomless BA and thus is complete for the Berman complexity class $\text{STA}(*, 2^{n^{O(1)}}, n)$ —problems solved by alternating exponential-time Turing machines with unrestricted space and n alternations—*i.e.* the same complexity as the first-order theory over the reals $\langle \mathbb{R}, +, 0, 1 \rangle$ with addition but no multiplication [4]. However, this result is restrictive in the sense that the formula class only contains $\{\bullet, \circ\}$ as constants, whereas in practice it is desirable to permit arbitrary tree constants, *e.g.* $\exists a \exists b. a \sqcup b = \widehat{\bullet \circ}$.

When the multiplication operator \bowtie is incorporated, the computational nature of the language becomes harder. The structure $\langle \mathbb{T}, \bowtie \rangle$ —without the Boolean operators—is isomorphic to word equations [27]. Accordingly, its first-order theory is undecidable while its existential theory is decidable with continuously improved complexity bounds currently at PSPACE and NP-hard (starting from Makanin’s argument [29] in 1977 and continuing with *e.g.* [22]).

Inspired by the notion of “semiautomatic structures” [21], Le *et al.* [27] restricted \bowtie to take only constants on the right-hand side, *i.e.* to a family of unary operators indexed by constants $\bowtie_{\tau} (x) \stackrel{\text{def}}{=} x \bowtie \tau$. Le *et al.* then examined $\mathcal{C} \stackrel{\text{def}}{=} \langle \mathbb{T}, \sqcup, \sqcap, \bar{\cdot}, \bowtie_{\tau} \rangle$. Note that the verification-sourced sentence (2) from §2.3 fits perfectly into \mathcal{C} : $\forall \pi. \pi = \bowtie_{\bullet} \widehat{\circ} (\pi) \oplus \bowtie_{\circ} \widehat{\bullet} (\pi)$. Le *et al.* encoded \mathcal{C} into *tree-automatic structures* [6], *i.e.*, logical structures whose constants can be encoded as trees, and domains and predicates finitely represented by tree automata. As a result, its first-order theory—with arbitrary tree constants—is decidable [6, 5, 36], but until our results in §4 the true complexity of \mathcal{C} was unknown.

3 Complexity of Boolean structure $\mathcal{A} \stackrel{\text{def}}{=} \langle \mathbb{T}, \sqcup, \sqcap, \bar{\cdot} \rangle$

Existing tree share solvers [25, 28] only utilize the additive operator \oplus in certain restrictive first-order segments. Given the fact that \oplus is defined from the Boolean structure $\mathcal{A} = \langle \mathbb{T}, \sqcup, \sqcap, \bar{\cdot} \rangle$, it is compelling to establish the decidability and complexity results over the general structure \mathcal{A} . More importantly, operators in \mathcal{A} can help reduce the complexity of a given formula. For example, consider the following separation logic entailment:

$$a \xrightarrow{\tau} 1 \star a \xrightarrow{\widehat{\bullet \circ}} \vdash a \xrightarrow{\widehat{\bullet \circ}} 1 \star \top.$$

To check the above assertion, entailment solvers have to extract and verify the following corresponding tree share formula by grouping shares from same heap addresses using \oplus and then applying equality checks:

$$\forall \tau \forall \tau'. \tau \oplus \widehat{\bullet \bullet}_\circ = \tau' \rightarrow \exists \tau'' . \tau'' \oplus \widehat{\bullet \bullet}_\circ = \tau'.$$

By using Boolean operators, the above $\forall \exists$ formula can be simplified into a \forall formula by specifying that either the share in the antecedent is not possible, or the share in the consequent is a ‘sub-share’ of the share in the antecedent:

$$\forall \tau. \neg(\tau \sqcap \widehat{\bullet \bullet}_\circ = \circ) \vee (\widehat{\bullet \bullet}_\circ \sqsubseteq \tau \oplus \widehat{\bullet \bullet}_\circ).$$

where the ‘sub-share’ relation \sqsubseteq is defined using Boolean union:

$$a \sqsubseteq b \stackrel{\text{def}}{=} a \sqcup b = b.$$

In this section, we will prove the following precise complexity of \mathcal{A} :

Theorem 1. *The first-order theory of \mathcal{A} is \leq_{\log} -complete for $\text{STA}(*, 2^{n^{O(1)}}, n)$, even if we allow arbitrary tree constants in the formulae.*

One important implication of the above result is that the same complexity result still holds even if the additive operator \oplus is included into the structure:

Corollary 1. *The Boolean tree share structure with addition $\mathcal{A}_\oplus = \langle \mathbb{T}, \oplus, \sqcup, \sqcap, \bar{\cdot} \rangle$ is \leq_{\log} -complete for $\text{STA}(*, 2^{n^{O(1)}}, n)$, even with arbitrary tree constants in the formulae.*

Proof. Recall that \oplus can be defined in term of \sqcup and \sqcap without additional quantifier variable:

$$a \oplus b = c \stackrel{\text{def}}{=} a \sqcup b = c \wedge a \sqcap b = \circ.$$

As a result, one can transform, in linear time, any additive constraint into Boolean constraint using the above definition. Hence the result follows. \square

Theorem 1 is stronger than the result in [27] which proved the same complexity but for restricted tree share constants in the formulae:

Proposition 1 ([27]). *The first-order theory of \mathcal{A} , where tree share constants are $\{\bullet, \circ\}$, is \leq_{\log} -complete for $\text{STA}(*, 2^{n^{O(1)}}, n)$.*

The hardness proof for lower bound of Theorem 1 is obtained directly from Prop. 1. To show that the same complexity holds for upper bound, we construct an $O(n^2)$ algorithm **flatten** (Alg. 1) that transforms arbitrary tree share formula into an equivalent tree share formula whose constants are $\{\bullet, \circ\}$:

Lemma 1. *Suppose $\text{flatten}(\Phi) = \Phi'$. Then:*

Algorithm 1 Flattening a Boolean tree share formula

```
1: function flatten( $\Phi$ )
Require:  $\Phi$  is a Boolean tree sentence
Ensure: Return an equivalent formula of height zero
2:   if height( $\Phi$ ) = 0 then return  $\Phi$ 
3:   else
4:     let  $s$  be the shape of  $\Phi$ 
5:     for each atomic formula  $\Psi$  in  $\Phi$ :  $t^1 = t^2$  or  $t^1 \text{ op } t^2 = t^3$ ,  $\text{op} \in \{\sqcup, \sqcap\}$  do
6:        $[t_1^i, \dots, t_n^i] \leftarrow \text{split}(t^i, s)$  for  $i = 1 \dots n$   $\triangleright n$  is the number of leaves in  $s$ 
7:        $\Psi_i \leftarrow t_1^i = t_2^i$  or  $t_1^i \text{ op } t_2^i = t_3^i$  for  $i = 1 \dots n$ 
8:        $\Phi \leftarrow$  replace  $\Psi$  with  $\bigwedge_{i=1}^n \Psi_i$ 
9:     end for
10:    for each quantifier  $Qv$  in  $\Phi$  do
11:       $[v_1, \dots, v_n] \leftarrow \text{split}(v, s)$ 
12:       $\Phi \leftarrow$  replace  $Qv$  with  $Qv_1 \dots Qv_n$ 
13:    end for
14:    return  $\Phi$ 
15:  end if
16: end function
17:
18: function split( $t, s$ )
Require:  $t$  is either a variable or a constant,  $s$  is a shape
Ensure: Return a list of decomposing components of  $t$  according to shape  $s$ 
19:   if  $s = *$  then return  $[t]$ 
20:   else let  $s = \widehat{s_0 s_1}$  in
21:     if  $t$  is  $\bullet$  or  $\circ$  then return concat(split( $t, s_0$ ), split( $t, s_1$ ))
22:     else if let  $t = \widehat{t_1 t_2}$  in then return concat(split( $t_0, s_0$ ), split( $t_1, s_1$ ))
23:     elseif  $t$  is a variable return concat(split( $t_0, s_0$ ), split( $t_1, s_1$ ))
24:     end if
25:   end if
26: end function
```

1. Φ' only contains $\{\bullet, \circ\}$ as constants.
2. Φ and Φ' have the same number of quantifier alternations.
3. Φ and Φ' are equivalent with respect to \mathcal{A} .
4. **flatten** is $O(n^2)$. In particular, if the size of Φ is n then Φ' has size $O(n^2)$.

Proof of Theorem 1. The lower bound follows from Prop. 1. By Lemma 1, we can use **flatten** in Alg. 1 to transform a tree formula Φ into an equivalent formula Φ' of size $O(n^2)$ that only contains $\{\bullet, \circ\}$ as constants and has the same number of quantifier alternations as in Φ . By Prop. 1, Φ' can be solved in $\text{STA}(*, 2^{n^{O(1)}}, n)$. This proves the upper bound and thus the result follows. \square

It remains to prove the correctness of Lemma 1. But first, we will provide a descriptive explanation for the control flow of **flatten** in Alg. 1. On line 2, it checks whether the height of Φ , which is defined to be the height of the highest tree constant in Φ , is zero. If it is the case then no further computation is needed as Φ only contains $\{\bullet, \circ\}$ as constants. Otherwise, the shape s (Defini-

tion 1) is computed on line 4 to guide the subsequent decompositions. On lines 5-9, each atomic sub-formula Ψ is decomposed into sub-components according to the shape s by the function `split` described on lines 18-26. Intuitively, `split` decomposes a tree τ into subtrees (line 21-22) or a variables v into new variables with appropriate binary subscripts (line 23). On line 8, the formula Ψ is replaced with the conjunction of its sub-components $\bigwedge_{i=1}^n \Psi_i$. Next, each quantifier variable Qv in Φ is also replaced with a sequence of quantifier variables $Qv_1 \dots Qv_n$ (lines 10-13). Finally, the modified formula Φ is returned as the result on line 14. The following example demonstrates the algorithm in action:

Example 1. Let $\Phi : \forall a \exists b. a \sqcup b = \widehat{\bullet \circ} \vee \neg(\bar{a} = \widehat{\circ \bullet})$. Then $\text{height}(\Phi) = 2 > 0$ and its shape s is $\widehat{\ast \ast \ast}$. Also, Φ contains the following atomic sub-formulae:

$$\Psi : a \sqcup b = \widehat{\bullet \circ} \quad \text{and} \quad \Psi' : \bar{a} = \widehat{\circ \bullet}.$$

After applying the `split` function to Ψ and Ψ' with shape s , we acquire the following components:

1. $\Psi_1 : a_{00} \sqcup b_{00} = \bullet, \Psi_2 : a_{01} \sqcup b_{01} = \circ, \Psi_3 : a_{10} \sqcup b_{10} = \circ, \Psi_4 : a_{11} \sqcup b_{11} = \circ.$
2. $\Psi'_1 : \bar{a}_{00} = \circ, \Psi'_2 : \bar{a}_{01} = \circ, \Psi'_3 : \bar{a}_{10} = \bullet, \Psi'_4 : \bar{a}_{11} = \circ.$

The following result formula is obtained by replacing Ψ with $\bigwedge_{i=1}^4 \Psi_i$, Ψ' with $\bigwedge_{i=1}^4 \Psi'_i$, $\forall a$ with $\forall a_{00} \forall a_{01} \forall a_{10} \forall a_{11}$, and $\exists b$ with $\exists b_{00} \exists b_{01} \exists b_{10} \exists b_{11}$:

$$\forall a_{00} \forall a_{01} \forall a_{10} \forall a_{11} \exists b_{00} \exists b_{01} \exists b_{10} \exists b_{11}. \bigwedge_{i=1}^4 \Psi_i \vee \neg \left(\bigwedge_{i=1}^4 \Psi'_i \right).$$

Definition 1 (Tree shape). A shape of a tree τ , denoted by $\langle \tau \rangle$, is obtained by replacing its leaves with \ast , e.g. $\langle \widehat{\bullet \circ} \rangle = \widehat{\ast \ast}$. The combined shape $s_1 \sqcup s_2$ is defined by overlapping s_1 and s_2 , e.g. $\widehat{\ast \ast} \sqcup \widehat{\ast \ast} = \widehat{\ast \ast \ast}$. The shape of a formula Φ , denoted by $\langle \Phi \rangle$, is the combined shape of its tree constants and \ast .

Note that tree shapes are not canonical, otherwise all shapes are collapsed into a single shape \ast . We are now ready to prove the first three claims of Lemma 1:

Proof of Lemma 1.1, 1.2 and 1.3. Observe that the shape of each atomic sub-formula Ψ is ‘smaller’ than the shape of Φ , i.e. $\langle \Psi \rangle \sqcup \langle \Phi \rangle = \langle \Phi \rangle$. As a result, each formula in the decomposition of `split`(Ψ , $\langle \Phi \rangle$) always has height zero, i.e. its only constants are $\{\bullet, \circ\}$. This proves claim 1.

Next, recall that the number of quantifier alternations is the number of times where quantifiers are switched from \forall to \exists or vice versa. The only place that `flatten` modifies quantifiers is on line 12 in which the invariant for quantifier alternations is preserved. As a result, claim 2 is also justified.

We are left with the claim that `flatten` is $O(n^2)$ where n is the size of the input formula Φ . By a simple analysis of `flatten`, it is essentially equivalent

to show that the result formula has size $O(n^2)$. First, observe that the formula shape $\langle \Phi \rangle$ has size $O(n)$ and thus we need $O(n)$ decompositions for each atomic sub-formula Ψ and each quantifier variable Qv of Φ . Also, each component in the decomposition of Ψ (or Qv) has size at most the size of Ψ (or Qv). As a result, the size of the formula Φ' only increases by a factor of $O(n)$ compared to the size of Φ . Hence Φ' has size $O(n^2)$. \square

To prove claim 4, we first establish the following result about the `split` function. Intuitively, this lemma asserts that one can use `split` together with some tree shape s to construct an isomorphic Boolean structure whose elements are lists of tree shares:

Lemma 2. Let $\text{split}_s \stackrel{\text{def}}{=} \lambda\tau. \text{split}(\tau, s)$, e.g. $\text{split}_{\widehat{\circ}}(\widehat{\circ}) = [\widehat{\circ}, \bullet, \bullet]$.

Then split_s is an isomorphism from \mathcal{A} to $\mathcal{A}' = \langle \mathbb{T}^n, \sqcup', \sqcap', \bar{\cdot}' \rangle$ where n is the number of leaves in s and each operator in \mathcal{M}' is defined component-wise from the corresponding operator in \mathcal{A} , e.g. $[a_1, a_2] \sqcup' [b_1, b_2] = [a_1 \sqcup a_2, b_1 \sqcup b_2]$.

Proof. W.l.o.g. we will only prove the case $s = \widehat{\circ}$ as similar argument can be obtained for the general case. By inductive arguments, we can prove that split_s is a bijection from \mathbb{T} to $\mathbb{T} \times \mathbb{T}$. Furthermore:

1. $\text{split}_s(a) \diamond \text{split}_s(b) = \text{split}_s(c)$ iff $a \diamond b = c$ for $\diamond \in \{\sqcup, \sqcap\}$.
2. $\text{split}_s(\bar{\tau}) = \overline{\text{split}_s(\tau)}$.

Hence split_s is an isomorphism from \mathcal{A} to $\mathcal{A}' = \langle \mathbb{T} \times \mathbb{T}, \sqcup', \sqcap', \bar{\cdot}' \rangle$. \square

Proof of Lemma 1.4. By Lemma 2, the function split_s allows us to transform formulae in \mathcal{A} into equivalent formulae over tree share lists in $\mathcal{A}' = \langle \mathbb{T}^n, \sqcup', \sqcap', \bar{\cdot}' \rangle$. On the other hand, observe that formulae in \mathcal{A}' can be rewritten into equivalent formulae in \mathcal{A} using conjunctions and extra quantifier variables, e.g. $\exists a \forall b. a \sqcup' b = [\widehat{\circ}, \bullet, \bullet]$ is equivalent to $\exists a_1 \exists a_2 \forall b_1 \forall b_2. a_1 \sqcup b_1 = \widehat{\circ} \wedge a_2 \sqcup b_2 = \bullet$. Hence the result follows. \square

The correctness of Lemma 1 is now fully justified. We end this section by pointing out a refined complexity result for the existential theory of \mathcal{A} , which corresponds to the satisfiability problem of quantifier-free formulae. Note that the number of quantifier alternations for this fragment is zero, and thus Theorem 1 only gives us an upper bound $\text{STA}(*, 2^{n^{O(1)}})$, which is exponential time complexity. Instead, we can use Lemma 1 to acquire the precise complexity:

Corollary 2. *The existential theory of \mathcal{A} , with arbitrary tree share constants, is NP-complete.*

Proof. Recall a classic result that existential theory of Countably Atomless BAs is NP-complete [30]. As \mathcal{A} belongs to this class, the lower bound is justified. To see why the upper bound holds, we use the function `flatten` to transform the input formula into standard BA formula and thus the result follows from Lemma 1. \square

4 Complexity of combined structure $\mathcal{C} \stackrel{\text{def}}{=} \langle \mathbb{T}, \sqcup, \sqcap, \bar{\cdot}, \bowtie_{\tau} \rangle$

In addition to the Boolean operators in §3, recall from §2.2 that tree shares also possess a multiplicative operator \bowtie that resembles the multiplication of rational permissions. As mentioned in §2.4, [27] showed that \bowtie is isomorphic to string concatenation, implying that the first-order theory of $\langle \mathbb{T}, \bowtie \rangle$ is undecidable, and so of course the first-order theory of $\langle \mathbb{T}, \sqcup, \sqcap, \bar{\cdot}, \bowtie \rangle$ is likewise undecidable.

By restricting multiplication to have only constants on the right-hand side, however, *i.e.* to the family of unary operators $\bowtie_{\tau}(x) \stackrel{\text{def}}{=} x \bowtie \tau$, Le *et al.* showed that decidability of the first-order theory was restored for the combined structure $\mathcal{C} \stackrel{\text{def}}{=} \langle \mathbb{T}, \sqcup, \sqcap, \bar{\cdot}, \bowtie_{\tau} \rangle$. However, Le *et al.* were not able to specify any particular complexity class. In this section, we fill in this blank by proving that the first-order theory of \mathcal{C} is nonelementary, *i.e.* that it cannot be solved by any resource-bound (space or time) algorithm:

Theorem 2. *The first-order theory of \mathcal{C} is non-elementary.*

To prove Theorem 2, we reduce the binary string structure with prefix relation [11], which is known to be nonelementary, into \mathcal{C} . Here we recall the definition and complexity result of binary strings structure:

Proposition 2 ([11, 35]). *Let $\mathcal{K} = \langle \{0, 1\}^*, S_0, S_1, \preceq \rangle$ be the binary string structure in which $\{0, 1\}^*$ is the set of binary strings, S_i is the successor function s.t. $S_i(s) = s \cdot i$, and \preceq is the binary prefix relation s.t. $x \preceq y$ iff there exists z satisfies $x \cdot z = y$. Then the first-order theory of \mathcal{K} is non-elementary.*

Before going into the technical detail, we briefly explain the many-one reduction from \mathcal{K} into \mathcal{C} . The key idea is that the set of binary strings $\{0, 1\}^*$ can be bijectively mapped into the set of *unary trees* $\mathcal{U}(\mathbb{T})$, trees that have exactly one black leaf, *e.g.* $\{\bullet, \widehat{\bullet}_o, \widehat{\bullet}_o \widehat{\bullet}_o, \widehat{\bullet}_o \widehat{\bullet}_o \widehat{\bullet}_o, \dots\}$. For convenience, we use the symbol \mathcal{L} to represent the left tree $\widehat{\bullet}_o$ and \mathcal{R} for the right tree \widehat{o}_\bullet . Then:

Lemma 3. *Let g map $\langle \{0, 1\}^*, S_0, S_1, \preceq \rangle$ into $\langle \mathbb{T}, \sqcup, \sqcap, \bar{\cdot}, \bowtie_{\tau} \rangle$ such that:*

1. $g(\epsilon) = \bullet, g(0) = \mathcal{L}, g(1) = \mathcal{R}$.
2. $g(b_1 \dots b_n) = g(b_1) \bowtie \dots \bowtie g(b_n), b_i \in \{0, 1\}$.
3. $g(S_0) = \lambda s. \bowtie_{\mathcal{L}}(g(s)), g(S_1) = \lambda s. \bowtie_{\mathcal{R}}(g(s))$.
4. $g(x \preceq y) = g(y) \sqsubseteq g(x)$ where $\tau_1 \sqsubseteq \tau_2 \stackrel{\text{def}}{=} \tau_1 \sqcup \tau_2 = \tau_2$.

Then g is a bijection from $\{0, 1\}^$ to $\mathcal{U}(\mathbb{T})$, and $x \preceq y$ iff $g(y) \sqsubseteq g(x)$.*

Proof. The routine proof that g is bijective is done by induction on the string length. Intuitively, the binary string s corresponds to the path from the tree root in $g(s)$ to its single black leaf, where 0 means ‘go left’ and 1 means ‘go right’. For example, the tree $g(110) = \mathcal{R} \bowtie \mathcal{R} \bowtie \mathcal{L} = \widehat{o}_\bullet \bowtie \widehat{o}_\bullet \bowtie \widehat{\bullet}_o = \widehat{o}_\bullet \widehat{\bullet}_o \widehat{\bullet}_o$ corresponds to the path right→right→left.

Now observe that if τ_1, τ_2 are unary trees then $\tau_1 \sqsubseteq \tau_2$ (i.e. the black-leaf path in τ_2 is a sub-path of the black-leaf path in τ_1) iff there exists a unary tree τ_3 such that $\tau_2 \bowtie \tau_3 = \tau_1$ (intuitively, τ_3 represents the difference path between τ_2 and τ_1). Thus $x \preceq y$ iff there exists z such that $xz = y$, iff $g(x) \bowtie g(z) = g(y)$, which is equivalent to $g(y) \sqsubseteq g(x)$ by the above observation. \square

In order for the reduction to work, we need to express the type of unary trees using operators from \mathcal{C} . The below lemma shows that the type of $\mathcal{U}(\mathbb{T})$ is expressible via a universal formula in \mathcal{C} :

Lemma 4. *A tree τ is unary iff it satisfies the following \forall -formula:*

$$\tau \neq \circ \wedge (\forall \tau'. \tau' \bowtie \mathcal{L} \sqsubseteq \tau \leftrightarrow \tau' \bowtie \mathcal{R} \sqsubseteq \tau).$$

$$\text{where } \tau_1 \sqsubseteq \tau_2 \stackrel{\text{def}}{=} \tau_1 \sqcup \tau_2 = \tau_2 \wedge \tau_1 \neq \tau_2.$$

Proof. The \Rightarrow direction is proved by induction on the height of τ . The key observation is that if $\tau_1 \bowtie \tau_2 \sqsubseteq \tau_3$ and τ_2, τ_3 are unary then τ_1 is also unary, $\tau_1 \sqsubseteq \tau_3$ and thus $\tau_1 \bowtie \bar{\tau}_2 \sqsubseteq \tau_3$. Note that both \mathcal{L}, \mathcal{R} are unary and $\bar{\mathcal{L}} = \mathcal{R}$, hence the result follows.

For \Leftarrow , assume τ is not unary. As $\tau \neq \circ$, it follows that τ contains at least two black leaves in its representation. Let τ_1 be the tree that represents the path to one of the black leaves in τ , we have $\tau_1 \sqsubseteq \tau$ and for any unary tree τ_2 , if $\tau_1 \sqsubseteq \tau_2$ then $\tau_2 \not\sqsubseteq \tau$. As τ_1 is unary, we can rewrite τ_1 as either $\tau'_1 \bowtie \mathcal{L}$ or $\tau'_1 \bowtie \mathcal{R}$ for some unary tree τ'_1 . The latter disjunction together with the equivalence in the premise give us both $\tau'_1 \bowtie \mathcal{L} \sqsubseteq \tau$ and $\tau'_1 \bowtie \mathcal{R} \sqsubseteq \tau$. Also, we have $\tau_1 \sqsubseteq \tau'_1$ and thus $\tau'_1 \not\sqsubseteq \tau$ by the aforementioned observation. Hence $\tau'_1 = \tau_1 \bowtie \bullet = \tau'_1 \bowtie (\mathcal{L} \sqcup \mathcal{R}) \sqsubseteq \tau$ which is a contradiction. \square

Proof of Theorem 2. We employ the reduction technique in [17] where formulae in \mathcal{K} are interpreted using the operators from \mathcal{C} . The interpretation of constants and operators is previously mentioned and justified in Lemma 3. We then replace each sub-formula $\exists x. \Phi$ with $\exists x. x \in \mathcal{U}(\mathbb{T}) \wedge \Phi$ and $\forall x. \Phi$ with $\forall x. x \in \mathcal{U}(\mathbb{T}) \rightarrow \Phi$ using the formula in Lemma 4. It follows that the first-order complexity of \mathcal{C} is bounded below by the first-order complexity of \mathcal{K} . Hence by Prop. 2, the first-order complexity of \mathcal{C} is nonelementary. \square

5 Causes of, and mitigants to, the nonelementary bound

Having proven the nonelementary lower bound for the combined theory in §4, we discuss causes and mitigants. In §5.1 we show that the nonelementary behavior of \mathcal{C} comes from the combination of both the additive and multiplicative theories by proving an elementary upper bound on a generalization of the multiplicative theory, and in §5.2 we discuss why we believe that verification tools in practice will avoid the nonelementary lower bound.

5.1 Complexity of multiplicative structure $\mathcal{B} \stackrel{\text{def}}{=} \langle \mathbb{T}, \tau \bowtie, \bowtie_\tau \rangle$

Since the first-order theory over $\langle \mathbb{T}, \bowtie \rangle$ is undecidable, it may seem plausible that the nonelementary behaviour of \mathcal{C} comes from the \bowtie_τ subtheory rather than the “simpler” Boolean subtheory \mathcal{A} , even though the specific proof of the lower bound given in §4 used both the additive and multiplicative theories (*e.g.* in Lemma 4). This intuition, however, is mistaken. In fact, even if we generalize the theory to allow multiplication by constants on either side—*i.e.*, by adding $\tau \bowtie(x) \stackrel{\text{def}}{=} \tau \bowtie x$ to the language—the restricted multiplicative theory $\mathcal{B} \stackrel{\text{def}}{=} \langle \mathbb{T}, \tau \bowtie, \bowtie_\tau \rangle$ is elementary. Specifically, we will prove that the first-order theory of \mathcal{B} is $\text{STA}(*, 2^{O(n)}, n)$ -complete and thus elementarily decidable:

Theorem 3. *The first-order theory of \mathcal{B} is $\leq_{\log\text{-lin}}$ -complete for $\text{STA}(*, 2^{O(n)}, n)$.*

Therefore, the nonelementary behavior of \mathcal{C} arises precisely because of the combination of both the additive and multiplicative subtheories.

We prove Theorem 3 by solving a similar problem in which two tree shares $\{\bullet, \circ\}$ are excluded from the tree domain \mathbb{T} . That is, let $\mathbb{T}^+ = \mathbb{T} \setminus \{\bullet, \circ\}$ and $\mathcal{B}^+ = \langle \mathbb{T}^+, \tau \bowtie, \bowtie_\tau \rangle$, we want:

Lemma 5. *The complexity of $\text{Th}(\mathcal{B}^+)$ is $\leq_{\log\text{-lin}}$ -complete for $\text{STA}(*, 2^{O(n)}, n)$.*

By using Lemma 5, the proof for the main theorem is straightforward:

Proof of Theorem 3. The hardness proof is direct from the fact that membership constraint in \mathcal{B}^+ can be expressed using membership constraint in \mathcal{B} :

$$\tau \in \mathcal{B}^+ \quad \text{iff} \quad \tau \in \mathcal{B} \wedge \tau \neq \circ \wedge \tau \neq \bullet.$$

As a result, any sentence from \mathcal{B}^+ can be transformed into equivalent sentence in \mathcal{B} by rewriting each $\forall v. \Phi$ with $\forall v. (v \neq \circ \wedge v \neq \bullet) \rightarrow \Phi$ and each $\exists v. \Phi$ with $\exists v. v \neq \circ \wedge v \neq \bullet \wedge \Phi$.

To prove the upper bound, we use the guessing technique as in [27]. In detail, we partition the domain \mathbb{T} into three disjoint sets:

$$S_1 = \{\circ\} \quad S_2 = \{\bullet\} \quad S_3 = \mathbb{T}^+.$$

Suppose the input formula contains n variables, we then use a ternary vector of length n to guess the partition domain of these variables, *e.g.*, if a variable v is guessed with the value $i \in \{1, 2, 3\}$ then v is assigned to the domain S_i . In particular, if v is assigned to S_1 or S_2 , we substitute v for \circ or \bullet respectively. Next, each bowtie term $\bowtie_\tau(a)$ or $\tau \bowtie(a)$ that contains tree share constants \bullet or \circ is simplified using the following identities:

$$\tau \bowtie \bullet = \bullet \bowtie \tau = \tau \quad \tau \bowtie \circ = \circ \bowtie \tau = \circ.$$

After this step, all the atomic sub-formulae that contain \circ or \bullet are reduced into either variable equalities $v_1 = v_2, v = \tau$ or trivial constant equalities such as $\bullet = \bullet, \widehat{\bullet}_\circ = \circ$ that can be replaced by either \top or \perp . As a result, the

new equivalent formula is free of tree share constants $\{\bullet, \circ\}$ whilst all variables are quantified over the domain \mathbb{T}^+ . Such formula can be solved using the Turing machine that decides $\text{Th}(\mathcal{B}^+)$. The whole guessing process can be integrated into the alternating Turing machine without increasing the formula size or number of quantifiers (*i.e.* the alternating Turing machine only needs to make two extra guesses \bullet and \circ for each variable and the simplification only takes linear time). Hence this justifies the upper bound. \square

The rest of this section is dedicated to the proof of Lemma 5. To prove the complexity $\text{Th}(\mathcal{B}^+)$, we construct an efficient isomorphism from \mathcal{B}^+ to the structure of ternary strings in $\{0, 1, 2\}^*$ with prefix and suffix successors. The existence of such isomorphism will ensure the complexity matching between the tree structure and the string structure. Here we recall a result from [34] about the first-order complexity of the string structure with successors:

Proposition 3 ([34]). *Let $\mathcal{S} = \langle \{0, 1\}^*, P_0, P_1, S_0, S_1 \rangle$ be the structure of binary strings with prefix successors P_0, P_1 and suffix successors S_0, S_1 such that:*

$$P_0(s) = 0 \cdot s \quad P_1(s) = 1 \cdot s \quad S_0(s) = s \cdot 0 \quad S_1(s) = s \cdot 1.$$

Then the first-order theory of \mathcal{S} is $\leq_{\log\text{-lin}}$ -complete for $\text{STA}(, 2^{O(n)}, n)$.*

The above result cannot be used immediately to prove our main theorem. Instead, we use it to infer a more general result where successors are not only restricted to 0 and 1, but also allowed to be any string s in a finite alphabet:

Lemma 6. *Let Σ be a finite alphabet of size $k \geq 2$ and $\mathcal{S}' = \langle \Sigma^*, P_s, S_s \rangle$ the structure of k -ary strings with infinitely many prefix successors P_s and suffix successors S_s where $s \in \Sigma^*$ such that:*

$$P_s(s') = s \cdot s' \quad S_s(s') = s' \cdot s.$$

Then the first-order theory of \mathcal{S}' is $\leq_{\log\text{-lin}}$ -complete for $\text{STA}(, 2^{O(n)}, n)$.*

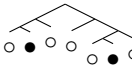
Proof. Although the proof in [34] only considers binary alphabet, the same result still holds even for finite alphabet Σ of size $k \geq 2$ with k prefix and suffix successors. Let $s = a_1 \dots a_n$ where $a_i \in \Sigma$, the successors P_s and S_s can be defined in linear size from successors in \mathcal{S} as follows:

$$P_s \stackrel{\text{def}}{=} \lambda s'. P_{a_1}(\dots P_{a_n}(s')) \quad S_s \stackrel{\text{def}}{=} \lambda s'. S_{a_n}(\dots S_{a_1}(s')).$$

These definitions are quantifier-free and thus the result follows. \square

Next, we recall some key results from [27] that establishes the fundamental connection between trees and strings in word equation:

Proposition 4 ([27]). *We call a tree τ in \mathbb{T}^+ prime if $\tau = \tau_1 \bowtie \tau_2$ implies either $\tau_1 = \bullet$ or $\tau_2 = \bullet$. Then for each tree τ in \mathbb{T}^+ , there exists a unique sequence of prime trees $\{\tau_i\}_{i=1}^n$ such that $\tau = \tau_1 \bowtie \dots \bowtie \tau_n$. As a result, each tree in \mathbb{T}^+ can be treated as a string in a word equation in which the alphabet is \mathbb{P} , the countably infinite set of prime trees, and \bowtie is the string concatenation.*

For example, the factorization of  is $\widehat{\bullet \circ} \bowtie \widehat{\bullet \circ} \bowtie \widehat{\circ \bullet}$, which is

unique. Prop. 4 asserts that by factorizing tree shares into prime trees, we can effectively transform multiplicative tree share constraints into equivalent word equations. Ideally, if we can represent each prime tree as a unique letter in the alphabet then Lemma 5 would follow from Lemma 6. Unfortunately, the set of prime trees \mathbb{P} are infinite [27] while Lemma 6 requires a finite alphabet. As a result, our tree encoding needs to be more sophisticated than the naïve way. The key observation here is that, as \mathbb{P} is countably infinite, there must be a bijective *encoding function* $I : \mathbb{P} \mapsto \{0, 1\}^*$ that encodes each prime tree into binary string, including the empty string ϵ . We need not to know the construction of I in advance, but it is important to keep in mind that I exists and the delay of its construction is intentional. We then extend I into \hat{I} that maps tree shares in \mathbb{T}^+ into ternary string in $\{0, 1, 2\}^*$ where the letter 2 purposely represents the delimiter between two consecutive prime trees:

Lemma 7. *Let $\hat{I} : \mathbb{T}^+ \mapsto \{0, 1, 2\}^*$ be the mapping from tree shares into ternary strings such that for prime trees $\tau_i \in \mathbb{P}$ where $i \in \{1, \dots, n\}$, we have:*

$$\hat{I}(\tau_1 \bowtie \dots \bowtie \tau_n) = I(\tau_1) \cdot 2 \dots 2 \cdot I(\tau_n).$$

By Prop. 4, \hat{I} is bijective. Furthermore, let $\tau_1, \tau_2 \in \mathbb{T}^+$ then:

$$\hat{I}(\tau_1 \bowtie \tau_2) = \hat{I}(\tau_1) \cdot 2 \cdot \hat{I}(\tau_2).$$

Having the core encoding function \hat{I} defined, it is now routine to establish the isomorphism from the tree structure \mathcal{B}^+ to the string structure \mathcal{S}' :

Lemma 8. *Let f be a function that maps the tree structure $\langle \mathbb{T}^+, \tau \bowtie, \bowtie_\tau \rangle$ into the string structure $\langle \{0, 1, 2\}, P_{s2}, S_{2s} \rangle$ such that:*

1. For each tree $\tau \in \mathbb{T}^+$, we let $f(\tau) \stackrel{\text{def}}{=} \hat{I}(\tau)$.
2. For each function $\tau \bowtie$, we let $f(\tau \bowtie) \stackrel{\text{def}}{=} P_{\hat{I}(\tau)2}$.
3. For each function \bowtie_τ , we let $f(\bowtie_\tau) \stackrel{\text{def}}{=} S_{2\hat{I}(\tau)}$.

Then f is an isomorphism from \mathcal{B}^+ to \mathcal{S}' .

Proof of Lemma 5. For the upper bound, observe that the function f in Lemma 8 can be used to transform tree share formulae in \mathcal{B}^+ to string formulae in \mathcal{S}' . It remains to ensure that the size of the string formula is not exponentially exploded. In particular, it suffices to construct \hat{I} such that if a tree $\tau \in \mathbb{T}^+$ has size n , its corresponding string $\hat{I}(\tau)$ has linear size $O(n)$. Recall that \hat{I} is extended from I which can be constructed in many different ways. Thus to avoid the size explosion, we choose to specify the encoding function I on the fly *after observing the input tree share formula*. To be precise, given a formula Φ in \mathcal{B} , we first factorize all its tree constants into prime trees, which can be done in log-space [27]. Suppose the formula has n prime trees $\{\tau_i\}_{i=1}^n$ sorted in

the ascending order of their sizes, we choose the most efficient binary encoding by letting $I(\tau_i) = s_i$ where s_i is the i^{th} string in length-lexicographic (shortlex) order of $\{0, 1\}^*$, *i.e.* $\{\epsilon, 0, 1, 00, 01, \dots\}$. This encoding ensures that the size of τ_i and the length of s_i only differ by a constant factor. Given the fact that a tree share in its factorized form $\tau_1 \bowtie \dots \bowtie \tau_n$ only requires $O(\sum_{i=1}^n \hat{I}(\tau_i))$ bits to represent, we infer that its size and the length of its string counterpart $\hat{I}(\tau)$ also differ by a constant factor. Hence, the upper bound complexity is justified.

To prove the lower bound, we need to construct the inverse function f^{-1} that maps the string structure \mathcal{S}' into the tree share structure \mathcal{B} . Although the existence of f^{-1} is guaranteed since f is isomorphism, we also need to take care of the size explosion problem. It boils down to construct an efficient mapping I^{-1} from binary strings to prime trees by observing the input string formula Φ . For each string constant $s_1 2 \dots 2 s_n$ in Φ where $s_i \in \{0, 1\}^*$, we extract all of the binary strings s_i . We then maps each distinct binary string s_i to a unique prime tree τ_i as follows. Let $k(0) = \widehat{\bullet}_\circ$, $k(1) = \widehat{\circ}_\bullet$ and assume $s_i = a_0 \dots a_m$ for $a_i \in \{0, 1\}$, we compute $\tau = k(a_0) \bowtie \dots \bowtie k(a_m)$. Then the mapped tree share for the string s_i is constructed as $\tau_i = \widehat{\bullet}_\tau$ (if $s_i = \epsilon$ then $\tau_i = \widehat{\bullet}_\circ$). It follows that τ_i is prime and this skewed tree has size $O(n)$ where n is the length of s_i . Thus the result follows. \square

Example 2. Consider the tree formula $\forall a \exists b \exists c. a = b \bowtie \widehat{\circ}_\bullet \circ \wedge b = \widehat{\circ}_\bullet \circ \bowtie c$. This formula contains two constants whose factorizations are below:

$$c_1 = \widehat{\circ}_\bullet \circ = \widehat{\bullet}_\circ \bowtie \widehat{\circ}_\bullet \quad c_2 = \widehat{\circ}_\bullet \circ = \widehat{\circ}_\bullet \bowtie \widehat{\bullet}_\circ.$$

We choose I such that $I(\widehat{\bullet}_\circ) = \epsilon$ and $I(\widehat{\circ}_\bullet) = 0$. Our encoding gives $s_1 = 20$ and $s_2 = 02$. This results in the string formula $\forall a \exists b \exists c. a = S_{220}(b) \wedge b = P_{022}(c)$ whose explicit form is $\forall a \exists b \exists c. a = b220 \wedge b = 022c$.

Now suppose that we want to transform the above string formula into equivalent tree formula. Following the proof of Lemma 5, we extract from the formula two binary strings $s_1 = \epsilon$ and $s_2 = 0$ which are mapped to the prime trees $\tau_1 = \widehat{\bullet}_\circ$ and $\tau_2 = \widehat{\circ}_\bullet$ respectively. Hence the equivalent tree share formula is $\forall a \exists b \exists c. a = \bowtie_{\tau_1 \bowtie \tau_2} (b) \wedge b = \tau_2 \bowtie_{\tau_1} (c)$. It is worth noticing the difference between this tree formula and the original tree formula, which suggests the fact that the representation of the alphabet (*i.e.* prime trees) is not important.

5.2 Combined \mathcal{C} formulae in practice

The source of the nonelementary behavior comes from two factors. First, as proven just above, it comes from the combination of both the additive and multiplicative operations of tree shares. Second, it comes from the number of quantifier alternations in the formula being analyzed, due to the encoding of \mathcal{C} in tree automata [27] and the resulting upper bound (the transformed automata of first-order formulae of tree automatic structures have sizes bounded by a tower of exponentials whose height is the number of quantifier alternations [6, 5]).

Happily, in typical verifications, especially in highly-automated verifications such as those done by tools like HIP/SLEEK [28], the number of quantifier alternations in formulae is small, even when carrying out complex verifications or inference. For example, consider the following biabduction problem (a separation-logic-based inference procedure) handled by the ShareInfer tool from [26]:

$$a \xrightarrow{\pi} (b, c, d) \star \widehat{\bullet} \circ \cdot \pi \cdot \text{tree}(c) \star \widehat{\bullet} \circ \cdot \pi \cdot \text{tree}(d) \star [??] \vdash \widehat{\bullet} \circ \cdot \pi \cdot \text{tree}(a) \star [??]$$

ShareInfer will calculate $\widehat{\bullet} \circ \cdot \pi \cdot \text{tree}(d)$ for the antiframe and $a \xrightarrow{\pi \bowtie \widehat{\bullet} \circ} (b, c, d) \star \widehat{\bullet} \circ \cdot \pi \cdot \text{tree}(d)$ for the inference frame. Although these guesses are a bit sophisticated, verifying them depends on [16] the following quantifier-alternation-free \mathcal{C} sentence: $\forall \pi, \pi'. \pi = \pi' \Rightarrow \bowtie \widehat{\bullet} \circ (\pi) \oplus \bowtie \widehat{\bullet} \circ (\pi) = \pi'$. Even with complex loop invariants, more than one alternation would be surprising because *e.g.* verification tools tend to maintain formulae in well-chosen canonical forms.

Moreover, because tree automata are closely connected to other well-studied domains, we can take advantage of existing tools such as MONA [23]. As an experiment we have hand-translated \mathcal{C} formulae into WS2S, the language of MONA, using the techniques of [10]. The technical details of the translation are provided in appendix §A. For the above formula, MONA reported 205 DAG hits and 145 nodes, with essentially a 0ms running time.

Lastly, heuristics are well-justified both because of the restricted problem formats we expect in practice as well as because of the nonelementary worst-case lower bound we proved in §4, opening the door to newer techniques like antichain/simulation [1].

6 Future work and conclusion

We have developed a tighter understanding of the complexity of the tree share model. As Boolean Algebras, their first-order theory is $\text{STA}(*, 2^{n^{O(1)}}, n)$ -complete, even with arbitrary tree constants in the formulas. Although the first-order theory over tree multiplication is undecidable [27], we have found that by restricting multiplication to be by a constant (on both the left \bowtie and right \bowtie_τ sides) we obtain a substructure \mathcal{B} whose first-order theory is $\text{STA}(*, 2^{O(n)}, n)$ -complete. Accordingly, we have two structures whose first-order theory has elementary complexity. Interestingly, their combined theory is still decidable but nonelementary, even if we only allow multiplication by a constant on the right \bowtie_τ .

We have several directions for future work. It is natural to investigate the precise complexity of the existential theory with the Boolean operators and right-sided multiplication \bowtie_τ (structure \mathcal{C}). The encoding into tree-automatic structures from [27] provides only an exponential-time upper bound (because of the result for the corresponding fragment in tree-automatic structures, *e.g.*, see [36]), and there is the obvious NP lower bound that comes from propositional logic satisfiability. We do not know if the Boolean operators ($\sqcup, \sqcap, \bar{\cdot}$) in combination

with the left-sided multiplication \bowtie_{\exists} is decidable (existential or first order, with or without the right-sided multiplication $\bowtie_{\exists r}$). Determining if the existential theory with the Boolean operators and *unrestricted* multiplication \bowtie is decidable also seems challenging. We would also like to know if the monadic second-order theory over these structures is decidable.

Acknowledgement. We would like to thank anonymous referees for their constructive reviews. Le and Lin are partially supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement no 759969). Le and Hobor are partially supported under Yale-NUS College grant R-607-265-322-121.

References

1. Parosh Aziz Abdulla, Yu-Fang Chen, Lukás Holík, Richard Mayr, and Tomás Vojnar. When simulation meets antichains. In *TACAS*, pages 158–174, 2010.
2. A. W. Appel, R. Dockins, A. Hobor, L. Beringer, J. Dodds, G. Stewart, S. Blazy, and X. Leroy. *Program Logics for Certified Compilers*. Cambridge U. Press, 2014.
3. Andrew W. Appel, Robert Dockins, and Aquinas Hobor. Mechanized semantic library, 2009.
4. Leonard Berman. The complexity of logical theories. *Theoretical Computer Science*, 11(1):71–77, May 1980.
5. A. Blumensath. *Automatic Structures*. PhD thesis, RWTH Aachen, 1999.
6. A. Blumensath and E. Grade. Finite presentations of infinite structures: automata and interpretations. In *Theory of Computer Systems*, pages 641–674, 2004.
7. Richard Bornat, Cristiano Calcagno, Peter O’H, and Matthew Parkinson. Permission accounting in separation logic. In *POPL*, pages 259–270, 2005.
8. John Boyland. Checking interference with fractional permissions. In *SAS*, pages 55–72, 2003.
9. Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.
10. Thomas Colcombet and Christof Löding. Transforming structures by set interpretations. *Logical Methods in Computer Science*, 3(2), 2007.
11. K. J. Compton and C. W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *APAL 1990*.
12. The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.
13. Robert Dockins, Aquinas Hobor, and Andrew W. Appel. A fresh look at separation algebras and share accounting. In *APLAS*, pages 161–177, 2009.
14. J. Dohrau, A. J. Summers, C. Urban, S. Münger, and P. Müller. Permission inference for array programs. In *CAV*, 2018.
15. Marko Doko and Viktor Vafeiadis. Tackling real-life relaxed concurrency with FSL++. In *ESOP*, pages 448–475, 2017.
16. Cristian A. Gherghina. *Efficiently Verifying Programs with Rich Control Flows*. PhD thesis, National University of Singapore, 2012.
17. Erich Grädel. Simple interpretations among complicated theories. *Information Processing Letters*, 35(5):235 – 238, 1990.
18. Hobor and Cristian Gherghina. Barriers in concurrent separation logic. In *ESOP*, pages 276–296, 2011.

19. Aquinas Hobor. *Oracle Semantics*. PhD thesis, Princeton University, Department of Computer Science, Princeton, NJ, October 2008.
20. Aquinas Hobor and Cristian Gherghina. Barriers in concurrent separation logic: Now with tool support! *Logical Methods in Computer Science*, 8(2), 2012.
21. Sanjay Jain, Bakhadyr Khoushainov, Frank Stephan, Dan Teng, and Siyuan Zou. Semiautomatic structures. In *CSR*, pages 204–217, 2014.
22. Artur Jez. Recompression: A simple and powerful technique for word equations. *J. ACM*, 63(1):4:1–4:51, 2016.
23. Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual*. BRICS, Department of Computer Science, Aarhus University, January 2001.
24. Duy-Khanh Le, Wei-Ngan Chin, and Yong Meng Teo. Threads as resource for concurrency verification. In *PEPM*, pages 73–84, 2015.
25. Xuan-Bach Le, Cristian Gherghina, and Aquinas Hobor. Decision procedures over sophisticated fractional permissions. In *APLAS*, 2012.
26. Xuan-Bach Le and Aquinas Hobor. Logical reasoning over disjoint fractional permissions. In *ESOP*, 2018.
27. Xuan-Bach Le, Aquinas Hobor, and Anthony W. Lin. Decidability and complexity of tree shares formulas. In *FSTTCS*, 2016.
28. Xuan-Bach Le, Thanh-Toan Nguyen, Aquinas Hobor, and Wei-Ngan Chin. A certified decision procedure for tree shares. In *ICFEM*, 2017.
29. G. S Makanin. The problem of solvability of equations in a free semigroup. In *Mat. Sbornik*, pages 147–236, 1977.
30. Kim Marriott and Martin Odersky. Negative boolean constraints. In *Theoretical Computer Science 160*, pages 365–380, 1996.
31. Peter W. O’Hearn. Resources, concurrency, and local reasoning. *Theor. Comput. Sci.*, 375(1-3):271–307, April 2007.
32. Matthew Parkinson. *Local Reasoning for Java*. PhD thesis, University of Cambridge, 2005.
33. Matthew J. Parkinson, Richard Bornat, and Peter W. O’Hearn. Modular verification of a non-blocking stack. In *POPL 2007*, pages 297–302, 2007.
34. Tatiana Rybina and Andrei Voronkov. Upper bounds for a theory of queues. In *ICALP*, pages 714–724, 2003.
35. L. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, M.I.T., 1974.
36. A. W. To. *Model Checking Infinite-State Systems: Generic and Specific Approaches*. PhD thesis, LFCS, School of Informatics, University of Edinburgh, 2010.
37. Jules Villard. *Heaps and Hops*. Ph.D. thesis, Laboratoire Spécification et Vérification, École Normale Supérieure de Cachan, France, February 2011.
38. T.D. Young, P.R. Pinto, K.J. Andersen, and L. Birkedal. Caper: Automatic verification for fine-grained concurrency. In *ESOP 2017*, 2017.

A Appendix

Fig. 1 contains the MONA WS2S encoding of the following tree share formula

$$\forall \pi, \pi'. \pi = \pi' \Rightarrow (\pi \bowtie \circ \widehat{\bullet}) \oplus (\pi \bowtie \bullet \widehat{\circ}) = \pi'.$$

where lower case letters are for variables of binary strings and upper case letters are for second-order monadic predicates. The last three lines in the code

```

ws2s;
pred ant(var2 Y) =
  all1 x,y: (x~y & x in Y & y in Y) => (~(x<=y) & ~(y<=x));
pred maxt(var2 X,var2 Y) =
  X sub Y & ex1 r:all1 x: x in X =>
  (r <= x & all1 z: r <= z => ex1 x': x' in X & (z <= x' | x' <= z));
pred roott(var1 x,var2 X) =
  all1 y: y in X & x <= y & all1 z:all1 y':y' in X & z <= y' => x <= z;
pred subt(var2 X, var2 Y) =
  all1 x1:all2 X':(maxt(X',X) & roott(x1,X')) =>
  (ex2 Y':maxt(Y',Y) => roott(x1,Y'));
pred eqt(var2 X, var2 Y) =
  subt(X,Y) & subt(Y,X);
pred singleton(var2 X) =
  ex1 x: x in X & (all1 y: y in X => x = y);
pred uniont(var2 X,var2 Y,var2 Z) =
  Z = X union Y & empty(X inter Y);
pred mint(var2 X) =
  all2 Y: maxt(Y,X) => singleton(Y);
pred sub0(var2 X, var2 X0) =
  all1 x:x in X <=> x.0 in X0;
pred sub1(var2 X, var2 X0) =
  all1 x:x in X <=> x.1 in X0;
pred leftMul(var2 X,var2 X') =
  all2 Y:(eqt(X,Y) & mint(Y)) => sub0(Y,X');
pred rightMul(var2 X,var2 X') =
  all2 Y:(eqt(X,Y) & mint(Y)) => sub1(Y,X');

all2 X,X',XL,XR,XU:
  (ant(X) & ant(X') & ant(XL) & ant(XR) & ant(XU) & eqt(X,X') &
  leftMul(X,XL) & rightMul(X,XR) & uniont(XL,XR,XU)) => (eqt(XU,X'));

```

Fig. 1. The transformation of tree share formula in §5.2 into equivalent WS2S formula.

are the formulas with a number of macros defined in the previous lines. Essentially, each tree share is represented by a second-order variable whose elements are *antichains* that describes a single path to one of its black leaves. Roughly speaking, the `eqt` predicate checks whether two tree shares are equal, `leftMul` and `rightMul` correspond to the multiplicative predicates $\bowtie_{\bullet \circ}^{\wedge}$ and $\bowtie_{\circ \bullet}^{\wedge}$ respectively, and `uniont` computes the additive operator \oplus . Other additional predicates are necessary for the consistent representation of the tree shares. In detail, `singleton(X)` means that X has exactly one element, `ant` makes sure any two antichains in the same tree are neither prefix of the other, `maxt(X,Y)` enforces that X is the maximal antichain of Y , `roott(x,X)` asserts x is the root of X , `subt` is a subset-like relation between two trees, while `mint` specifies the canonical form. Lastly, we have `sub0` and `sub1` as the intermediate predicates for the multiplicative predicates.