A Certified Decision Procedure for Tree Shares (Extended version)

Xuan-Bach Le*, Thanh-Toan Nguyen*, Wei-Ngan Chin*, and Aquinas Hobor[+*]

*School of Computing and +Yale-NUS College, National University of Singapore

Abstract. We develop a certified decision procedure for reasoning about systems of equations over the "tree share" fractional permission model of Dockins *et al.* Fractional permissions can reason about shared ownership of resources, *e.g.* in a concurrent program. We imported our certified procedure into the HIP/SLEEK verification system and found bugs in both the previous, uncertified, decision procedure and HIP/SLEEK itself. In addition to being certified, our new procedure improves previous work by correctly handling negative clauses and enjoys better performance.

1 Introduction

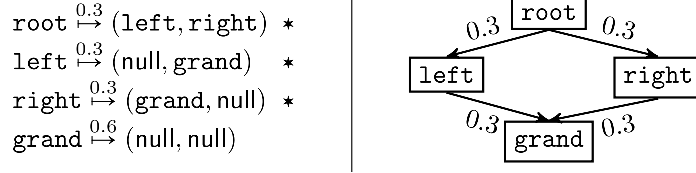
The last decade has enjoyed much progress in formal methods for concurrency in both theoretical understanding [?,?,?,?,?] and tool support [?,?,?,?,?,?,?]. Fractional shares enable reasoning about shared ownership of resources between multiple parties, *e.g.* in a concurrent program [?]. The original model for fractional shares was rational numbers in $[0, 1]$, with 0 representing no ownership, 1 representing full ownership, and $0 < x < 1$ representing partial ownership. A *policy* maps permission quanta to allowed actions. One simple policy maps 1 to the ability to both read and write a memory cell, $0 < x < 1$ to the ability to read—but not write—the cell, and 0 denying both reading and writing. We can prevent dangerous read/write and write/write data races by enforcing that the combined total ownership of each address is no more than 1.

Unfortunately, rational numbers are not an ideal model for shares. Consider the following recursive predicate definition for fractionally-owned binary trees:

$$\begin{aligned} \mathsf{tree}(\ell, \pi) \ &\stackrel{\text{def}}{=} \ (\ell = \mathsf{null} \ \wedge \mathsf{emp}) \ \vee \\ &\exists \ell_l, \ell_r.\ (\ell \stackrel{\pi}{\mapsto} (\ell_l, \ell_r) \star \mathsf{tree}(\ell_l, \pi) \star \mathsf{tree}(\ell_r, \pi)) \end{aligned} \tag{1}$$

Here we write $a \stackrel{\pi}{\mapsto} b$ to indicate that memory location a contains value b and is owned with (positive/nonempty) share π. We can split and join ownership of a cell with addition: $a \stackrel{\pi_1}{\mapsto} b \star a \stackrel{\pi_2}{\mapsto} b \dashv\vdash a \stackrel{\pi_1 \oplus \pi_2}{\longmapsto} b$; note we use $\oplus$ instead of $+$ to indicate that the addition is bounded in [0,1] and thus partial (*e.g.* $0.6 \oplus 0.6$ is undefined). This tree predicate is obtained directly from the standard recursive predicate for binary trees in separation logic by asserting only π ownership of the root and recursively doing the same for the left and right substructures, and so at

first glance looks obviously correct. The problem is that when $\pi \in (0, 0.5]$, then tree can describe some non-tree directed acyclic graphs such as the following:



This heap satisfies $\mathsf{tree}(\mathtt{root}, 0.3)$ despite actually being a DAG ($\mathtt{grand}$ is owned with share $0.3 \oplus 0.3 = 0.6$).

Parkinson proposed a model based on sets of natural numbers that solved this issue but introduced others [**?**], and then Dockins *et al.* [**?**] proposed the following "tree share" model, which fixes all of the aforementioned issues. A tree share $\tau \in \mathbb{T}$ is inductively defined as a binary tree with boolean leaves:

$$\tau \triangleq \circ \mid \bullet \mid \widehat{\tau\ \tau}$$

Here $\circ$ denotes an "empty" leaf while $\bullet$ a "full" leaf. The tree $\circ$ is thus the empty share, and $\bullet$ the full share. There are two "half" shares: $\widehat{\circ\ \bullet}$ and $\widehat{\bullet\ \circ}$, and four "quarter" shares, beginning with $\widehat{\widehat{\bullet\ \circ}\ \circ}$. It is a feature, rather than a bug, that the two half shares are distinct from each other.

Notice that we presented the first quarter share as $\widehat{\widehat{\bullet\ \circ}\ \circ}$ instead of *e.g.* $\widehat{\widehat{\bullet\ \circ}\ \widehat{\circ\ \circ}}$. This is deliberate: the second choice is not a valid share because the tree is not in *canonical form*. A tree is in canonical form when it is in its most compact representation under the relation $\cong$:

$$\frac{}{\circ \cong \circ} \qquad \frac{}{\bullet \cong \bullet} \qquad \frac{}{\circ \cong \widehat{\circ\ \circ}} \qquad \frac{}{\bullet \cong \widehat{\bullet\ \bullet}} \qquad \frac{\tau_1 \cong \tau_1' \qquad \tau_2 \cong \tau_2'}{\widehat{\tau_1\ \tau_2} \cong \widehat{\tau_1'\ \tau_2'}}$$

Maintaining canonical form is a headache in Coq but does not introduce any fundamental difficulty. Accordingly, for this presentation we will simply fold and unfold trees to/from canonical form when required by the narrative.

Defining the "join" operation $\oplus$ on tree shares formally is somewhat technical due to the necessity of managing the canonical forms [**?**, §A] but the core idea is quite straightforward. Simply unfold both trees under $\cong$ into the same shape and join them leafwise using the rules $\circ \oplus \circ = \circ$, $\circ \oplus \bullet = \bullet$, and $\bullet \oplus \circ = \bullet$; afterwards refold under $\cong$ back into canonical form. Here is an example:

$$\widehat{\widehat{\bullet\ \circ}\ \circ} \oplus \widehat{\widehat{\circ\ \bullet}\ \widehat{\bullet\ \circ}} \cong \widehat{\widehat{\bullet\ \circ}\ \widehat{\circ\ \circ}} \oplus \widehat{\widehat{\circ\ \bullet}\ \widehat{\bullet\ \circ}} = \widehat{\widehat{\bullet\ \bullet}\ \widehat{\bullet\ \circ}} \cong \widehat{\bullet\ \widehat{\bullet\ \circ}}$$

Because $\bullet \oplus \bullet$ is undefined, the join relation on trees is a partial operation. Dockins *et al.* [**?**] prove that the join relation satisfies a number of useful axioms *e.g.* associativity and commutativity (§**??** has the full list). One key axiom, not satisfied by $(\mathbb{Q}, \oplus)$, is "disjointness": $x \oplus x = y \Rightarrow x = \circ$. Disjointness is the axiom that forces the tree predicate—equation **??**—to behave properly: we saw above that we get a DAG in $\mathbb{Q}$ since $x \oplus x$ need not be 0.

Due to their good metatheorical properties, various program logics [?,?] and tools [?,?,?] incorporate tree shares. Gherghina detailed a number of programs whose verifications used tree shares heavily [?, Ch.4]; these form the core of our benchmark in §??. However, most tools have avoided using tree shares in part because they lacked algorithms that could decide entailments involving fractionals. Hobor and Gherghina [?] showed how to divide an entailment between separation logic formulae incorporating fractional ownership into 1) a fraction-free separation logic entailment, and 2) an entailment between systems of share equations; this encouraged shares to be studied as a standalone domain.

Le *et al.* developed a tool to decide tree share entailments [?]. The present paper improves on their work in several ways. From a practical point of view, our new tool is fully machine-checked in Coq, giving the highest level of assurance that both its implementation and underlying theory are rock solid. By comparing our new tool with Le *et al.*'s, we discovered weaknesses in both the latter's implementation and its theory. Moreover, a trend in recent years has been to develop verification toolsets within Coq [?,?,?]; since certified tools generally only depend on other certified tools, such tools have not been able to use Le *et al.*'s implementation, but they can use our new tool. Happily, despite the challenges involved in developing an implementation in Coq, our new tool exhibits improved performance over Le *et al.*'s due to a number of heuristics that meaningfully improve performance without sacrificing soundness or completeness; some of these heuristics should be applicable to future certified procedures.

From a theoretical point of view, our major improvement over Le *et al.* is a sound treatment of negations. Negative clauses in logic are often more difficult to handle than positive ones are. Le *et al.*'s theory purported to support a very limited form of negation, which allowed them to force variables to be nonempty, *i.e.* $\pi \neq \circ$. We believe the previous theory is unsound when there are a sufficiently high number of nonempty variables on both sides of an implication. Our new theory handles arbitrary negative clauses, *i.e.* $\neg(\pi_1 \oplus \pi_2 = \pi_3)$ and is fully mechanized in Coq. A second theoretical improvement is a more careful treatment of existential variables.

The rest of this paper is organized as follows. In §?? we define the central decision problem and give an overview of our procedure. In §?? we show the key algorithms and outline why they are correct. All our proofs are mechanized in Coq; additional pen-and-paper details are also available in our appendix [?]. In §?? we discuss our 38.6k LOC certified implementation, describe how we have incorporated it into the HIP/SLEEK verification toolset [?], and benchmark its performance. Finally, in §?? we discuss related and future work and conclude.

2 Share constraints and their decision procedures

In §??, we introduce the decision problems over tree shares, satisfiability and entailment over *share equation systems.* Next we overview our decision procedure in §?? together with a brief description of their components' functionality. For **convenience**, we will use the symbol $\mathcal{L}$ to represent $\widehat{\bullet\ \circ}$ and $\mathcal{R}$ for $\widehat{\circ\ \bullet}$.

2.1 Share constraints

Given a SL entailment $P \vdash Q$ with fractional permissions, there are standard procedures to separately extract a heap constraint and a share constraint [?,?,?]. For example, the entailment $x \xmapsto{v_1} 1 \star y \xmapsto{\mathcal{R}} 2 \vdash x \xmapsto{\mathcal{L}} 1$ yields constraints $v_1 \neq \circ \wedge v_2 = \mathcal{R} \vdash \exists v_3.\ \mathcal{L} \oplus v_3 = v_1$. Tree constraints pose a technical difficulty due to the infinite tree domain, *e.g.*, $v_1 \oplus v_2 = \bullet$ has infinitely many solutions $\{(\bullet, \circ), (\mathcal{L}, \mathcal{R}), \ldots\}$. The type of tree constraints we need to deal with can be represent as $\Sigma_1 \vdash \Sigma_2$ where Σ_i is *share equation system*:

Definition 1. *A share equation system Σ is a quadruple $(l^\exists, l^=, l^+, l^-)$ in which:*

1. *$l^\exists$ is the list of existential variables.*
2. *$l^=$ is the list of equalities $\pi_1 = \pi_2$.*
3. *l^+ is the list of equations $\pi_1 \oplus \pi_2 = \pi_3$.*
4. *l^- is the list of disequations $\neg(\pi_1 \oplus \pi_2 = \pi_3)$.*

The entailment $\Sigma_1 \vdash \Sigma_2$ can be informally understood as "all solutions of Σ_1 are also solutions of Σ_2". In theory, it is conventional to treat equalities $\pi_1 = \pi_2$ as macros for $\pi_1 \oplus \circ = \pi_2$, although our certified tool tracks equalities separately for optimization purposes. For **convenience**, we will usually illustrate equation system as $\Sigma = \{x_1, \ldots, x_n, g_1, \ldots, g_m\}$ in which x_i is existential variable and g_i is either equality, equation or disequation.

To define the semantics of Σ, let *context* ρ be a mapping from variable names to tree shares. We then override ρ over tree constants as identity, *i.e.*, $\rho(\tau) = \tau$. To handle existential variable lists, we define the notion of a *context override*:

$$\rho[\rho' \Leftarrow l] \stackrel{\text{def}}{=} \lambda x.\ \rho'(v) \text{ if } x \in l \text{ else } \rho(v)$$

The semantics of forcing, written $\rho \models \Phi$, follows natural, *e.g.*, $\rho \models \pi_1 \oplus \pi_2 = \pi_3$ iff $\rho(\pi_1) \oplus \rho(\pi_2) = \rho(\pi_3)$ and $\rho \models P \wedge Q$ iff $\rho \models P$ and $\rho \models Q$. We say ρ is a solution of Σ, denoted by $\rho \models \Sigma$, if there exists a context ρ' such that $\rho[\rho' \Leftarrow l^\exists] \models l^= \wedge l^+ \wedge l^-$. Consequently, we say Σ_1 entails Σ_2 if all solutions of Σ_1 are also solutions of Σ_2. In this paper, we propose *certified algorithms* to solve the satisfiability and entailment over tree shares:

Problem. Let Σ_1, Σ_2 be share equation systems. Construct a sound and complete procedure to handle the following queries:

1. **SAT**(Σ_1): Is Σ_1 satisfiable, *i.e.*, $\exists \rho.\ \rho \models \Sigma_1$?
2. **IMP**(Σ_1, Σ_2): Does Σ_1 entail Σ_2, *i.e.*, $\forall \rho.\ \rho \models \Sigma_1 \Rightarrow \rho \models \Sigma_2$?

Despite allowing negative clauses, entailment is not subsumed by satisfiability due to the quantifier alternation in the consequent. One interesting exercise is to examine the metatheoretical properties of tree shares described by Dockins et al. [?]; these are given in Figure **??**. Several of these are the standard properties of separation algebras [?], but others are part of what make the tree share model special. In particular, tree shares are one of the fractional permission models that simultaneously satisfy Disjointness (forces the `tree` predicate—equation **??**—to

Functional: $\quad x \oplus y = z_1 \quad \Rightarrow \quad x \oplus y = z_2 \quad \Rightarrow \quad z_1 = z_2$
Commutative: $\quad x \oplus y \quad = \quad y \oplus x$
Associative: $\quad x \oplus (y \oplus z) \quad = \quad (x \oplus y) \oplus z$
Cancellative: $\quad x_1 \oplus y = z \quad \Rightarrow \quad x_2 \oplus y = z \quad \Rightarrow \quad x_1 = x_2$
Unit: $\quad \exists u.\ \forall x.\ x \oplus u = x$
Disjointness: $\quad x \oplus x = y \quad \Rightarrow \quad x = y$
Cross split: $\quad a \oplus b = z \wedge c \oplus d = z \Rightarrow \exists ac, ad, bc, bd.$
$ac \oplus ad = a \wedge bc \oplus bd = b \wedge ac \oplus bc = c \wedge ad \oplus bd = d$

∀ a b c d ∃ ac bc ad bd

Infinite splitability: $\quad x \neq \circ \quad \Rightarrow \quad \exists x_1, x_2.\ x_1 \neq \circ \ \wedge\ x_2 \neq \circ \ \wedge\ x_1 \oplus x_2 = x$

Fig. 1. Properties of tree shares

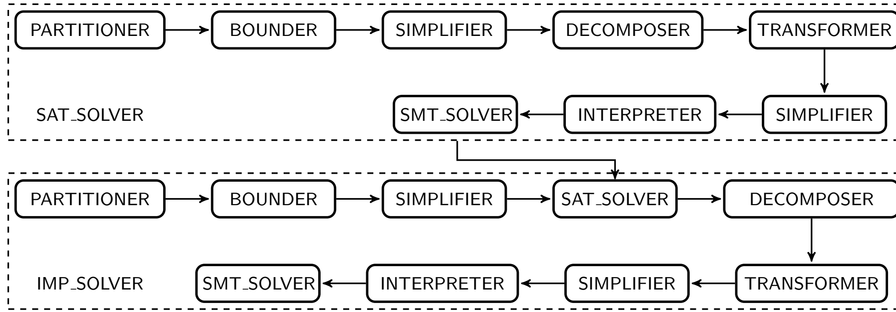


Fig. 2. SAT solver and IMP solver

behave properly), Cross-split (used *e.g.* in settings involving overlapping data structures), and Infinite splitability (to verify divide-and-conquer algorithms). Encouragingly, all of the properties except for "Unit" are expressible as entailments in our format; *e.g.* associativity is expressed as:

$$\{x \oplus a = b, y \oplus z = a\} \quad \vdash \quad \{c, x \oplus y = c, c \oplus z = b\}$$

Unit requires the order of quantifiers to swap; our format can express the weaker "Multiunit axiom" $\forall x.\ \exists u.\ x \oplus u = x$ as well as $\forall x.\ x \oplus \circ = x$.

2.2 Overview of our decision procedure

We use **SAT** and **IMP** for the problems and SAT and IMP for the decision procedures themselves. Although the entailment checker IMP is our main concern, the satisfiability checker SAT is helpful for at least two reasons. First, SAT helps to prune the search space; *e.g.*, if the antecedent Σ_1 for IMP is unsatisfiable, we can immediately conclude $\Sigma_1 \vdash \Sigma_2$. Second, the correctness of some of the transformations in IMP require that Σ_1 be satisfiable.

The architecture of our system is given in Figure **??**. We have two procedures to solve problems over share formulas, one for satisfiability and the other

for entailment, both written in Gallina and certified in Coq. Identically-named components in the two procedures are similar in spirit but not identical in operation; thus *e.g.* there are two different SIMPLIFIER components, one for SAT and another for IMP. The PARTITIONER, BOUNDER, and SIMPLIFIER components substantially improve the performance of our procedures in practice but are not complete solvers: in the worst case they do nothing. Since they are included for performance we will discuss them in more detail in §**??**.

The DECOMPOSER and TRANSFORMER components form the heart of our procedure. While the $\oplus$ operation has many useful properties that enable sophisticated reasoning about shared ownership in program verifications (*e.g.* Figure **??**), they are not strong enough for techniques like Gaussian elimination (which even in $\mathbb{Q}$ cannot handle negative clauses). In §**??** we will describe DECOMPOSER in detail after developing the necessary theory. Briefly, DECOMPOSER takes a system of equations with constants of arbitrary complexity and eventually produces a much larger equivalent system in which each constant is either $\circ$ or $\bullet$ (*i.e.*, the final system has *height* zero).

TRANSFORMER is a very sophisticated component mathematically, yet also the simplest computationally: it just changes the **type** of the system. That is, it inputs a **tree** system of height zero and outputs an equivalent, essentially identical **Boolean** system. The only actual computational content is by swapping $\circ$ for $\bot$ and $\bullet$ for $\top$. The join relation on Booleans is simply disjoint disjunction:

$$\top \oplus \bot = \top \qquad \bot \oplus \top = \top \qquad \bot \oplus \bot = \bot$$

The last option, $\top \oplus \top$, is undefined.

INTERPRETER translates Boolean systems of equations into Boolean sentences by rewriting positive and negative equations using the rules

$$\begin{aligned} \pi_1 \oplus \pi_2 = \pi_3 &\rightsquigarrow (\pi_1 \wedge \neg\pi_2 \wedge \pi_3) \vee (\neg\pi_1 \wedge \pi_2 \wedge \pi_3) \vee (\neg\pi_1 \wedge \neg\pi_2 \wedge \neg\pi_3) \\ \neg(\pi_1 \oplus \pi_2 = \pi_3) &\rightsquigarrow (\neg\pi_1 \vee \pi_2 \vee \neg\pi_3) \wedge (\pi_1 \vee \neg\pi_2 \vee \neg\pi_3) \wedge (\pi_1 \vee \pi_2 \vee \pi_3) \end{aligned}$$

Next, it adds the appropriate quantifiers depending on the query type to reach a closed sentence. INTERPRETER's code and correctness proof are straightforward.

SMT_SOLVER uses simple quantifier elimination to check the validity of boolean sentences. Our SMT solver is rather naïve, and thus is the performance bottleneck of our tool, but we could not find a suitable Gallina alternative. As discussed in §**??**, despite its naïveté our overall performance seems acceptable in practice due to the heuristics in PARTITION, BOUNDER, and SIMPLIFIER.

3 Core algorithms for the decision procedures

We begin with some basic definitions and notions in §**??** that are essential for the algorithms and their correctness proofs. In §**??** and §**??**, we propose our decision procedures to solve **SAT** and **IMP** together with illustrated examples.

Algorithm 1 Solver SAT for systems with disequations

```
1: function SAT(Σ)
2:     if SAT⁺(Σ⁺) = ⊥ then return ⊥
3:     else if l⁻ = nil then                         ▷ l⁻ is the disequation list in Σ
4:         return ⊤
5:     else   let l⁻ = [η_1, ..., η_n]
6:         return ⋀_{i=1}^{n} SSAT(Σ^{η_i})
```

3.1 Definitions and notations

We adopt the following definitions and notations. We use nil to denote empty list, $[e_1, \ldots, e_n]$ to represent list's content, and $l +\!\!+ l'$ for list concatenation. We use the metavariable η to represent a single disequation. The symbols Σ and Π are reserved for systems and pairs of systems respectively; if the exact form of our systems is not important or is clear from the context, we may refer it as Γ. The symbol ρ and S are for contexts and solutions respectively. We use $|\tau|$ to indicate the height of τ. Also, we will override the height function $|\cdot|$ for equation systems and contexts to indicate the height of the highest tree constant. For a tree τ, we let τ_l and τ_r to be the left and right sub-trees of τ, *i.e.*, $\tau = \tau_l = \tau_r$ if $\tau \in \{\circ, \bullet\}$ and $\tau = \widehat{\tau_l \; \tau_r}$ otherwise. We define several basic systems for SAT and IMP as the building blocks of the decision procedures:

Definition 2. *Let $\Sigma, \Sigma_1, \Sigma_2$ be share equation systems and η, η_1, η_2 disequations. Let l be a list of disequations, we define Σ^l to be the new equation system in which the disequation list in Σ is replaced with l. For convenience, we write Σ^η as shortcut for $\Sigma^{[\eta]}$, and Σ^+ as shortcut for Σ^{nil}. Then:*

1. *If the disequation list in Σ is empty then Σ is called a positive system.*
2. *If there is exactly one disequation in Σ then Σ is called a singleton system.*
3. *If Σ_1 is positive and Σ_2 is singleton then (Σ_1, Σ_2) is called a Z-system.*
4. *If both Σ_1 and Σ_2 are singleton then (Σ_1, Σ_2) is called a S-system.*

In particular, Σ^+ is always a positive system, Σ^η is always a singleton system, $(\Sigma_1^+, \Sigma_2^\eta)$ is always a Z-system, and $(\Sigma_1^{\eta_1}, \Sigma_2^{\eta_2})$ is always an S-system.

3.2 Decision procedure for SAT

We propose the procedure SAT (Alg. **??**) to solve **SAT** of systems with disequations. For $\mathbf{SAT}(\Sigma)$, the existential list is redundant and thus will be ignored. Our new decision procedure SAT also makes use of the old decision procedure SAT^+ from previous work [**?**] for systems without disequations, *e.g.*, positive systems. To help the readers gain intuition, we will abstract away all the tedious low-level implementations and only discuss about the high-level structure. The execution of SAT consists of two major steps which are described in Alg. **??**. First, the system Σ is separated into a list of singleton systems; each contains a single disequation taken from the disequation list of Σ. In the second step, each

Algorithm 2 Solver SSAT for singleton systems

1: **function** SSAT(Σ^η)
Require: Σ^η is singleton and Σ^+ is satisfiable
2: $[\Sigma_1, \dots, \Sigma_n] \leftarrow$ DECOMPOSE(Σ^η)
3: transform each Σ_i into Boolean formula Φ_i
4: $\Phi \leftarrow \bigvee_{i=1}^{n} \Phi_i$
5: **return** SMT_SOLVER(Φ)

Algorithm 3 Decompose system into sub-systems of height zero

1: **function** DECOMPOSE(Γ)
Require: Γ is either one system (**SAT**) or pair of systems (**IMP**)
Ensure: A list of sub-systems of height zero
2: **if** $|\Gamma| = 0$ **then return** $[\Gamma]$
3: **else**
4: $(\Gamma_1, \Gamma_2) \leftarrow$ SINGLE_DECOMPOSE(Γ)
5: **return** DECOMPOSE(Γ_1) ++ DECOMPOSE(Γ_2)
6:
7: **function** SINGLE_DECOMPOSE(Γ)
Require: Γ is either one system (**SAT**) or pair of systems (**IMP**)
Ensure: A pair of left and right sub-system
8: **if** $|\Gamma| = 0$ **then return** (Γ, Γ)
9: **else**
10: $\Gamma_l \leftarrow$ replace each tree constant τ in Γ with its left sub-tree τ_l
11: $\Gamma_r \leftarrow$ replace each tree constant τ in Γ with its right sub-tree τ_r
12: **return** (Γ_l, Γ_r)

singleton system is solved individually using the subroutine SSAT, then their results are conjoined to determine the result of SAT(Σ).

The solver SSAT for singleton system (Alg. **??**) calls subroutine DECOMPOSE (Alg. **??**) that helps decompose a share system into sub-systems of height zero. These subs-systems subsequently go though a 2-phase process to be transformed into Boolean formulas. In the first phase, the subroutine TRANSFORM trivially converts tree type into Boolean type using the conversions $\bullet \rightsquigarrow \top$ and $\circ \rightsquigarrow \bot$. Correspondingly, the share system is converted into the Boolean system. In the second phase, the subroutine INTERPRET helps to interpret the Boolean system into an equivalent Boolean formula by adding necessary quantifiers ($\exists$ for **SAT**, $\forall$ for **IMP**) and conjunctives among equations and disequations. Finally, Theorem **??** states the correctness of SAT whose proof is verified in Coq.

Theorem 1. *Let Σ be a share system then Σ is satisfiable iff* SAT(Σ) $= \top$.

Example 1. Let $\Sigma = \{v_1 \oplus v_2 = \bullet, \neg(v_1 = \mathcal{L}), \neg(v_2 = \circ)\}$ then **SAT**(Σ) is the valid formula ($v_1 = \mathcal{R}$, $v_2 = \mathcal{L}$ is a solution):

$$\exists v_1 \exists v_2.\ v_1 \oplus v_2 = \bullet \ \wedge\ \neg(v_1 = \mathcal{L}) \ \wedge\ \neg(v_2 = \circ)$$

Algorithm 4 Solver IMP for entailment of share systems with disequations

1: **function** $\mathsf{IMP}(\Sigma_1, \Sigma_2)$
2: **if** $\mathsf{SAT}(\Sigma_1) = \bot$ **then return** $\bot$
3: **else if** $\mathsf{IMP}^+(\Sigma_1^+, \Sigma_2^+) = \bot$ **then return** $\bot$
4: **else** let l_1^-, l_2^- be disequation lists of Σ_1, Σ_2
5: **if** $l_2^- = \mathsf{nil}$ **then return** $\top$
6: **else** let $l_2^- = [\eta_2^1, \ldots, \eta_2^n]$
7: **if** $l_1^- = \mathsf{nil}$ **then return** $\bigwedge_{i=1}^n \mathsf{ZIMP}(\Sigma_1^+, \Sigma_2^{\eta_2^i})$
8: **else** let $l_1^- = [\eta_1^1, \ldots, \eta_1^m]$
9: **for** $i = 1 \ldots n$ and $j = 1 \ldots m$ **do**
10: let $Z_i \leftarrow \mathsf{ZIMP}(\Sigma_1^+, \Sigma_2^{\eta_2^i})$ and $S_i^j \leftarrow \mathsf{SIMP}(\Sigma_1^{\eta_1^j}, \Sigma_2^{\eta_2^i})$
11: **return** $\bigwedge_{i=1}^n (Z_i \vee (\bigvee_{j=1}^m S_i^j))$

First, $\mathsf{SAT}^+(\Sigma^+)$ is called to check $\exists v_1 \exists v_2.\ v_1 \oplus v_2 = \bullet$ (which returns $\top$ as $v_1 = \circ$, $v_2 = \bullet$ is a solution). After that, Σ is split into two singleton systems:

$$\Sigma^1 = \{v_1 \oplus v_2 = \bullet, \neg(v_1 = \mathcal{L})\} \text{ and } \Sigma^2 = \{v_1 \oplus v_2 = \bullet, \neg(v_2 = \circ)\}$$

When $\mathsf{NSAT}(\Sigma^1)$ is called, Σ^1 is split into Σ_1^1 and Σ_2^1 by $\mathsf{DECOMPOSER}$:

$$\Sigma_1^1 = \{v_1 \oplus v_2 = \bullet, \neg(v_1 = \bullet)\} \text{ and } \Sigma_2^1 = \{v_1 \oplus v_2 = \bullet, \neg(v_1 = \circ)\}$$

The two systems Σ_1^1, Σ_2^1 are transformed into boolean formulas Φ_1^1 and Φ_2^1:

$$\begin{aligned} \Phi_1^1 &= \exists v_1 \exists v_2.\ ((v_1 \wedge \neg v_2) \vee (\neg v_1 \wedge v_2)) \wedge \neg v_1 \\ \Phi_2^1 &= \exists v_1 \exists v_2.\ ((v_1 \wedge \neg v_2) \vee (\neg v_1 \wedge v_2)) \wedge v_1 \end{aligned}$$

As both Φ_1^1 and Φ_2^1 are valid, $\mathsf{NSAT}(\Sigma^1)$ returns $\top$. Similarly, one can verify that $\mathsf{NSAT}(\Sigma^2)$ also returns $\top$ and thus $\mathsf{SAT}(\Sigma)$ returns $\top$ as the result. □

Additional details of the soundness proof for SAT can be found in [**?**, §B.1], which uses a technique we call "domain reduction", explained in [**?**, §A.1]. We finish §**??** by pointing out a decidability result of $\oplus$:

Corollary 1. *The $\exists$-theory of $\mathcal{M} = \langle \mathbb{T}, \oplus, = \rangle$ is decidable.*

Proof. Let Ψ be a quantifier-free formula in $\mathcal{M}$, we convert Ψ into Disjunctive Normal Form $\bigvee_{i=1}^n \Psi_i$ then each Ψ_i can be represented as a constraint system Σ_i. As a result, Ψ is satisfiable iff some Σ_i is satisfiable which can be solved using Algo. **??**. Thus the result follows.

3.3 Decision procedure for IMP

Our IMP procedure (Alg. **??**) deploys a similar strategy as for SAT by reducing the entailment into several entailments of the basic systems (*e.g.* Z-system and S-system). In detail, IMP verifies the entailment $\Sigma_1 \vdash \Sigma_2$ by first calling two

Algorithm 5 Solvers for entailment of Z-systems and S-systems

```
 1: function ZIMP(Σ1, Σ2)
Require: (Σ1, Σ2) is Z-system, Σ1 is satisfiable and Σ1 ⊢ Σ2+
 2:     [Γ1, ..., Γn] ← DECOMPOSE(Σ1, Σ2)
 3:     transform each Γi into Boolean formula Φi
 4:     Φ ← ⋁_{i=1}^n Φi
 5:     return SMT_SOLVER(Φ)
 6:
 7: function SIMP(Σ1, Σ2)
Require: (Σ1, Σ2) is S-system, Σ1+ is satisfiable, Σ1+ ⊢ Σ2+ and Σ1+ ⊬ Σ2
 8:     [Γ1, ..., Γn] ← DECOMPOSE(Σ1, Σ2)
 9:     transform each Γi into Boolean formula Φi
10:     Φ ← ⋀_{i=1}^n Φi
11:     return SMT_SOLVER(Φ)
```

solvers $\mathsf{SAT}(\Sigma_1)$ and $\mathsf{IMP}^+(\Sigma_1^+, \Sigma_2^+)$[1] (line 2 and 3). Then the lengths of the two disequation lists (l_1^- in Σ_1 and l_2^- in Σ_2) critically determine the subsequent flow of IMP. To be precise, there are three different cases of l_1^- and l_2^- that fully cover all the possibilities:

1. If $l_2^- = \mathsf{nil}$ (line 5) then the answer is equivalent to $\mathsf{IMP}^+(\Sigma_1^+, \Sigma_2^+)$, *i.e.*, $\top$.
2. Otherwise, we check whether $l_1^- = \mathsf{nil}$ (line 7) from which the answer is conjoined from several entailments of Z-systems $(\Sigma_1, \Sigma_2^{\eta_2^i})$; each is constructed from (Σ_1, Σ_2) by removing all disequations in Σ_2 except for one. Here we call the subroutine ZIMP which is a specialized for entailment of Z-systems.
3. The third case is neither l_1^- nor l_2^- is empty (line 8). Then $\Sigma_1 \vdash \Sigma_2$ is derived by taking the conjunction of several entailments of Z-systems and S-systems altogether. Here we use SIMP to solve S-system entailments.

Two specialized solvers ZIMP and SIMP are described in Alg. **??**. For ZIMP, we first call the subroutine DECOMPOSE to split the Z-system into sub-systems of height zero. Next, each sub-system is transformed in to Boolean formula by adding necessary quantifiers and logical connectives. These Boolean formulas are then combined using disjunctions to form a single Boolean formula; and this formula is solved using standard SMT solvers to determine the result of the entailment. The procedure for SIMP has a similar structure, except that the final Boolean formula is formed using conjunctions. Also, it is worth noticing that there are certain preconditions for both solvers; and all of them are important to shape the correctness of the solvers. Last but not least, the correctness of IMP is mentioned in Theorem **??**; and its proof is verified entirely in Coq.

Theorem 2. *Let* Σ_1, Σ_2 *be share systems then* $\Sigma_1 \vdash \Sigma_2$ *iff* $\mathsf{IMP}(\Sigma_1, \Sigma_2) = \top$.

Example 2. The infinite splitability of tree share (Fig.**??**):

$$\forall v.\ (v \neq \circ \Rightarrow \exists v_1 \exists v_2.\ v_1 \oplus v_2 = v \wedge v_1 \neq \circ \wedge v_2 \neq \circ)$$

[1] This is the entailment checker for positive constraints from previous work [**?**].

can be represented as the entailment $\Sigma_1 \vdash \Sigma_2$ s.t.:

$$\Sigma_1 = \{\neg(v = \circ)\} \text{ and } \Sigma_2 = \{v_1, v_2, v_1 \oplus v_2 = v, \neg(v_1 = \circ), \neg(v_2 = \circ)\}$$

This entailment will go though Algo. **??** until line 8 because both disequation lists are nonempty. As there are two disequations in Σ_2, namely $\eta_1 : v_1 \neq \circ$ and $\eta_2 : v_2 \neq \circ$, we need to verify the conjunction $P_1 \wedge P_2$ s.t.:

$$P_1 = \mathsf{ZIMP}(\Sigma_1^+, \Sigma_2^{\eta_1}) \vee \mathsf{SIMP}(\Sigma_1, \Sigma_2^{\eta_1}) \text{ and } P_2 = \mathsf{ZIMP}(\Sigma_1^+, \Sigma_2^{\eta_2}) \vee \mathsf{SIMP}(\Sigma_1, \Sigma_2^{\eta_2})$$

For P_1, $\mathsf{ZIMP}(\Sigma_1^+, \Sigma_2^{\eta_1})$ is equivalent to $\forall v.\ (\top \Rightarrow \exists v_1, v_2.\ v_1 \oplus v_2 = v \wedge v_1 \neq \circ)$ which is false by choosing $v = \circ$ so that both v_1 and v_2 must also be $\circ$. Likewise, $\mathsf{SIMP}(\Sigma_1, \Sigma_2^{\eta_1})$ is equivalent to $\forall v.\ (v \neq \circ \Rightarrow \exists v_1, v_2.\ v_1 \oplus v_2 = v \wedge v_1 \neq \circ)$ which is transformed into the boolean formula:

$$\Phi_1 \ = \ \forall v.\ (v \Rightarrow \exists v_1, v_2.\ ((\neg v_1 \wedge \neg v_2 \wedge \neg v) \vee (v_1 \wedge \neg v_2 \wedge v) \vee (\neg v_1 \wedge v_2 \wedge v)) \wedge v_1)$$

As Φ_1 is valid, P_1 is true. Same result holds for P_2 and thus $\Sigma_1 \vdash \Sigma_2$. □

Additional details of the soundness proof for IMP can be found in [**?**, §B.2], again using domain reduction [**?**, §A.1].

4 Performance, evaluation, and implementation

Having described the heart of our decision procedures, what remains is to describe the practical aspects of their development and evaluation. In §**??** we describe various techniques that enable good performance in practice. In §**??** we describe how we benchmarked our tool running inside Coq, running as a standalone compiled program, and after incorporating it into the HIP/SLEEK verification toolset. In [**?**, §C] we document the files in the development itself; we have approximately 38.6k lines of code in 31 files.

4.1 Performance-enhancing components

The architecture of our tool was given in §**??** (Figure **??**). The key DECOMPOSER, TRANSFORMER and INTERPRETER components were discussed in §**??**, §**??**, and §**??**. Here we give details on the PARTITIONER, BOUNDER, and SIMPLIFIER modules. Their principal goal is to shrink the search space and uncover contradictions, although they each do so in a very different way. Although in practice they can substantially improve performance, none of these components is a complete solver. The key ideas in these components were developed previously [**?**,**?**], although not all together. We have made a number of incremental enhancements, but our major contribution for these is components is the development of high-performing general-purpose certified implementations.

PARTITIONER. The goal of this module is to separate a constraint system into *independent subsystems*. Two systems are independent of each other if they do not share any common variable (with existential variables bound locally).

The partition function is implemented *generically*: in other words it does not assume very much about the underlying domain. To build the module, we must specify types of *variables* V, *equations* E, and contexts C. We also provide a function $\sigma : E \Rightarrow L(V)$ that extracts a list of variables from an equation, an overriding function written $\rho'[\rho \Leftarrow l]$, and an evaluation relation written $c \models e$. The soundness proof requires two properties that relate these inputs as follows:

$$\frac{\rho \models e \qquad \sigma(e) \cap l = \emptyset}{\rho[\rho' \Leftarrow l] \models e}\ \textit{disjointness} \qquad\qquad \frac{\rho \models e \qquad \sigma(e) \subset l}{\rho'[\rho \Leftarrow l] \models e}\ \textit{inclusion}$$

Disjointness and inclusion jointly specify that satisfaction of an equation only depends on the variables it contains: overriding variables not in the equation does not matter; and from any context, if we override all of the variables that are in an equation then we can ignore the original context.

It is simple to use PARTITIONER for **SAT**, but to handle **IMP** is harder. We can "tag" equations and variables as coming from the antecedent or consequent before partitioning and then use these tags to separate the resulting partitioned systems into antecedents and consequents afterwards.

The implementation of PARTITIONER is nontrivial in purely functional languages like Coq. One reason is that we need a purely functional union-find data structure, which we obtain via the impure-to-pure transformation of Pippenger [**?**] applied to the canonical imperative algorithm [**?**]. In other words, we substitute red-black trees for memory (mapping "addresses" to "cell contents") and pay a logarithmic access penalty, yielding an $O\big(n \cdot \mathsf{log}(n) \cdot \alpha(n)\big)$ algorithm.

The termination of "find" turns out to be subtle. Parent pointers are represented as cells that "point to" other cells; however, those parent cells can be anywhere in the red-black tree (*e.g.* item 5 can be the parent of item 10, or the other way around.) Accordingly an important invariant of the structure is that "nonlocal links" form acyclic chains, which is the key termination argument.

Given union-find, the algorithm is straightforward: each variable is put into a singleton set, and then while processing each equation we union the corresponding sets. Lastly, we extract the sets and filter the equations into components.

BOUNDER. The bounder uses order theory to prune the space. Each variable v is given an initial bound $\circ \subseteq v \subseteq \bullet$. The bounder then tries to narrow these bounds by forward and backward propagation. For example, if $\tau_1 \subseteq v_1 \subseteq \bullet$, $\tau_2 \subseteq v_2 \subseteq \bullet$, and $\circ \subseteq v_3 \subseteq \bullet$, then if $v_1 \oplus v_2 = v_3$ is an clause we can conclude that v_3's lower bound can be increased from $\circ$ to $\tau_1 \sqcup \tau_2$ (where $\sqcup$ computes the union in an underlying lattice on trees). In some cases, the bounds for a variable can be narrowed all the way to a point, in which case we can substitute the variable away. In other cases we can find a contradiction (when the upper bound goes below the lower bound), allowing us to terminate the procedure.

The bounder is an updated version of the incomplete solver developed by Hobor *et al.* [**?**]. Although our main contribution here is the certified implementation, we managed to tighten the bounds in certain cases.

SIMPLIFIER*.* The simplifier is a combination of a substitution engine and several effective heuristics for reducing the overall difficulty via calculation. For example, from $v \oplus \tau_1 = \tau_2$, where τ_i are constants, we can compute an exact value for v using an inverse of $\oplus$: $v = \tau_2 \ominus \tau_1$. SIMPLIFIER also hunts for contradictions: for example, from $v \oplus v = \bullet$ we can reach a contradiction due to the "disjointness" axiom from Figure **??**. The core idea of simplifier was contained in the work of Le *et al.* [**?**], so our main contribution here is our certified implementation.

4.2 Experimental evaluation

Our procedures are implemented and certified in Coq. Users who wish to use our code outside of Coq can use Coq's extraction feature to generate code in OCaml and Haskell, although at present a small bug in Coq 8.4pl5's extraction mechanism requires a small human edit to the generated code.

We benchmarked our code in three ways using an Intel i7 with 8GB RAM. First, we used a suite of 102 standalone test cases developed by Le *et al.* (53 **SAT** and 49 **IMP**) [**?**] and the 9 metatheoretic properties described in §**??**. These tests cover a variety tricky cases such as large number of variables, deep tree constants, etc. Even running as interpreted Gallina code within Coq, the time is extremely encouraging at **17 seconds** to check all 111 tests. After we port to Coq 8.5 we can use the native_compute tactic to increase performance.

Second, we compiled the extracted OCaml code with ocamlopt. The total running time to test all 111 previous tests is **0.02 seconds**, despite our naïve SMT solver; our previous tool took 1.4 seconds. Since our SMT solver is a separate module, it can be replaced with a more robust external solver such as Z3 [**?**] if performance is bottleneck in that spot in the future.

Finally, we incorporated our solver into the HIP/SLEEK verification toolset, which was previously using the uncertified solver by Le *et al.*. We did so by writing a short (approximately 150 line) "shim" that translated the format used by the previous tool into the format expected by the new tool.

We then benchmarked our tool against a suite of 23 benchmark programs as shown in Figure **??**. 15 of those programs were developed by Gherghina [**?**] and utilize a concurrent separation logic for pthreads-style barriers that exercise share provers extensively. Another 7 tests were developed for the HipCAP project [**?**], which extended HIP/SLEEK to reason in a Concurrent Abstract Predicate [**?**] style. Finally, we wrote a simple fork/join program for our initial testing.

The results are rather interesting! The left column gives the input file name to HIP/SLEEK and the second the number of lines in that file. The third column is the total number of calls into the solver (both **SAT** and **IMP**). **The fourth column is the number of times the previous solver by Le *et al.* answered the query incorrectly.** The fifth column gives the time (in seconds) spent by Le *et al.*'s uncertified solver and the sixth column gives the time spent

File	LOC	# calls	# wrong	Le *et al.* [?]	Our tool
MISD_ex1_th1.ss	36	294	48	2.21	2.37
MISD_ex1_th2.ss	36	495	67	4.36	4.48
MISD_ex1_th3.ss	36	726	94	6.95	6.58
MISD_ex1_th4.ss	36	1,003	123	9.09	8.36
MISD_ex1_th5.ss	36	1,320	134	15.74	12.38
MISD_ex2_th1.ss	47	837	107	16.77	18.97
MISD_ex2_th2.ss	52	1,044	157	29.34	26.02
MISD_ex2_th3.ss	87	1,841	260	69.09	64.21
MISD_ex2_th4.ss	105	3,023	374	194.17	194.64
PIPE_ex1_th2.ss	35	283	7	2.49	2.78
PIPE_ex1_th3.ss	44	467	12	4.92	4.65
PIPE_ex1_th4.ss	56	678	15	7.00	7.53
PIPE_ex1_th5.ss	66	931	18	9.67	9.37
SIMD_ex1_v2_th1.ss	74	1,167	281	18.46	17.64
SIMD_ex1_v2_th2.ss	95	2,029	392	63.83	53.50
cdl-ex1a-fm.ss	49	7	0	0.10	0.08
cdl-ex2-fm.ss	50	9	0	0.12	0.09
cdl-ex3-fm.ss	51	10	0	0.11	0.12
cdl-ex4-race.ss	50	5	0	0.09	0.09
cdl-ex4a-race.ss	50	9	0	0.10	0.08
cdl-ex5-deadlock.ss	42	5	0	0.10	0.10
cdl-ex5a-deadlock.ss	42	9	0	0.08	0.08
ex-fork-join.ss	25	47	22	0.19	0.16
total		**10,252**	**534**	**455.01**	**434.30**

Fig. 3. Evaluation of our procedures using HIP/SLEEK

by our new certified solver. HIP/SLEEK was benchmarked on a more powerful machine with 16 cores and 64GB RAM.

The uncertified solver got approximately 5.2% of the queries wrong! In our subsequent investigation, we discovered a number of bugs in the original solver: code rot (due to a change in the correct mechanism to call the SMT backend), improper error handling and signaling, general coding errors, and the incorrect treatment of nonzero variables. We also discovered bugs in HIP/SLEEK itself, which did not always use the result of the solver in the correct way; this is why the regression tests were passing even though the solver was reporting the incorrect answer. Our discovery of bugs on this scale, despite the large benchmarks developed by Le *et al.* [?] and Gherghina [?], illustrates the value of developing certified decision procedures.

Our timing results are reasonable: despite our naïve SMT solver backend and the difficulties in writing the algorithms in a purely functional style, our tool is approximately 4.6% faster than Le *et al.*'s uncertified solver.

5 Related work, future work, and conclusion

Boyland first proposed fractional shares over $\mathbb{Q}$ [?]. Subsequently, Bornat *et al.* [?] improved the rational model by adding natural counting permissions to reason about critical sections. Other notable refinements of the rationals are

achieved by Boyland *et al.* [**?**], Huisman *et al.* [**?**] and Müller *et al.* [**?**] that work well on programs with fork, join and lock. Parkinson showed that $\mathbb{Q}$'s lack of disjointness caused trouble and proposed modelling shares as subsets of $\mathbb{N}$ [**?**]. Dockins *et al.* proposed the tree share model used in the present paper to fix issues with Parkinson's model [**?**]. Hobor *et al.* were the first to use tree shares in a program logic [**?**], followed by Hobor and Gherghina [**?**] and Villard [**?**]. Hobor and Gherghina [**?**], Villiard [**?**], and Appel *et al.* [**?**] subsequently integrated shares into program verification tools with various incomplete solvers. Le *et al.* [**?**] developed sound and complete procedures to handle tree share constraints but their correctness proof only justifies the case when there is no disequation.

Future work. We have plans to examine the theory further to support general logical formulae (including arbitrary quantifier use) and perhaps monadic second-order logic. Dockins *et al.* also define a kind of multiplicative operation $\bowtie$ between shares whose computability and complexity was first analyzed by Le *et al.* [**?**]. Interestingly, this operator can be used to scale permissions over arbitrary predicates and thus our decision procedures need to be generalized to handle constraints that contain both $\oplus$ and $\bowtie$.

Conclusion. We have used tree shares to model permissions for integration into program logics. We proposed two decision procedures for tree shares and proved their correctness in Coq. The two algorithms perform well in practice and have been integrated into a sizable verification toolset.

A Formal definitions

The correctness of our decision procedures makes use of rounding functions and averaging function defined in [**?**]. We recall *left rounding* $\lfloor \overleftarrow{\tau} \rfloor_n$ (*right rounding* $\lfloor \overrightarrow{\tau} \rfloor_n$), which unfolds τ into full binary tree of height n, removes all left (right) leaves then folds it back to canonical form; and *averaging* $\tau_1 \bigtriangledown_n \tau_2$, which unfolds τ_1, τ_2 into full binary trees of height $n-1$, combines their leaves pairwise then folds the combined tree back to canonical form. Nevertheless, we introduce for clarity two new functions: *pair rounding* $\lfloor \tau \rceil_n = (\lfloor \overleftarrow{\tau} \rfloor_n, \lfloor \overrightarrow{\tau} \rfloor_n)$ and *combine* $\phi^{-1}(\tau_1, \tau_2)$ that merge two trees τ_1, τ_2 into $\widehat{\tau_1 \ \tau_2}$.

We illustrate these functions in Figure **??** to help readers gain intuition of these functions' mechanism. Our example is the tree at height 3 together with its derived trees. To help track what is going on in $\lfloor \overleftarrow{\tau} \rfloor_n$, $\lfloor \overrightarrow{\tau} \rfloor_n$, and $\tau_1 \bigtriangledown_{\tau_2} n$ we have highlighted the left leaf in each pair with the color red and the right leaf in each pair with the color blue. For $\phi(\tau)$ and $\phi^{-1}(\tau_1, \tau_2)$, we color red for leaves from the left subtree and blue for leaves from the right subtree.

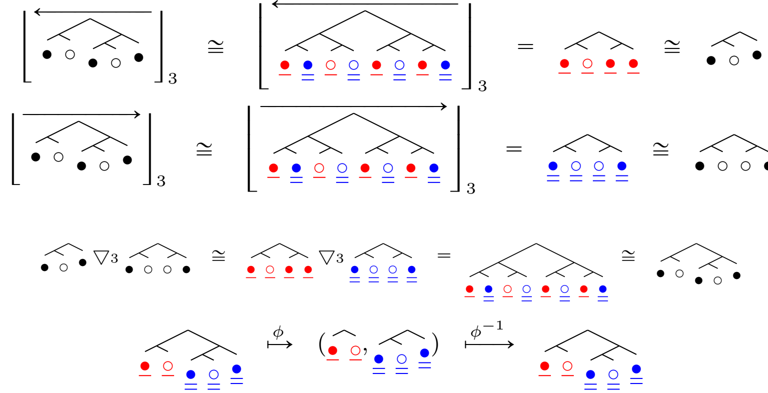


Fig. 4. Illustrated examples of applying the tree operators

Lemma 1. *Let $\mathbb{T}_n$ be the set of tree shares of height at most n. We override the join $\oplus$ over the domain $\mathbb{T}_n \times \mathbb{T}_m$ by applying the normal join component-wise,* i.e.*:*

$$(\tau_1, \tau_2) \oplus (\tau_1', \tau_2') \stackrel{def}{=} (\tau_1 \oplus \tau_1', \tau_2 \oplus \tau_2')$$

Then both the pair rounding function $\lfloor \cdot \rceil_{n+1} : \langle \mathbb{T}_{n+1}, \oplus \rangle \mapsto \langle \mathbb{T}_n \times \mathbb{T}_n, \oplus \rangle$ and split function $\phi : \langle \mathbb{T}_{n+1}, \oplus \rangle \mapsto \langle \mathbb{T}_n \times \mathbb{T}_n, \oplus \rangle$ are isomorphism, namely they are both bijection $\mathbb{T}_{n+1} \mapsto \mathbb{T}_n \times \mathbb{T}_n$ and they preserve the join $\oplus$:

$$\tau_1 \oplus \tau_2 = \tau_3 \ \ \textit{iff} \ \ \phi(\tau_1) \oplus \phi(\tau_2) = \phi(\tau_3) \ \ \textit{iff} \ \ \lfloor \tau_1 \rceil_{n+1} \oplus \lfloor \tau_2 \rceil_{n+1} = \lfloor \tau_3 \rceil_{n+1}$$

Furthermore:

1. *If* $|\tau| \leqslant n$ *then* $\lfloor\ \tau\ \rfloor_{n+1} = (\tau, \tau)$.
2. *If* $|\tau| = n + 1$ *then* $|\lfloor\overleftarrow{\tau}\rfloor_{n+1}| < |\tau|$ *and* $|\lfloor\overrightarrow{\tau}\rfloor_{n+1}| < |\tau|$.
3. *Let* $\phi(\tau) = (\tau_l, \tau_r)$. *If* $|\tau| = 0$ *then* $\tau_l = \tau_r = \tau$, *otherwise* $|\tau_l| < |\tau|$ *and* $|\tau_r| < |\tau|$.
4. ∇_{n+1} *is the inverse function of* $\lfloor\cdot\rfloor_{n+1}$.

Proof. Proved in Coq. By induction on the tree height.

Although both the pair rounding function $\lfloor\cdot\rfloor_n$ and split function ϕ are isomorphism, they are critically different in term of mechanism. Briefly speaking, the function $\lfloor\cdot\rfloor_n$ only affects trees of height n whereas the function ϕ affects all trees except $\bullet$ and $\circ$. As a result, $\lfloor\cdot\rfloor_n$ helps us 'refine' big solution into smaller one while ϕ^{-1} is used to decompose big system into smaller ones:

Corollary 2. *Let* ρ *be a context of* Σ *then:*

1. *If* $n \geqslant |\rho| > |\Sigma|$ *and* $\lfloor\rho\rfloor_n = (\rho_l, \rho_r)$ *then* $\rho \models \Sigma$ *iff both* $\rho_l \models \Sigma$ *and* $\rho_r \models \Sigma$.
2. *If* $n > |\rho|$ *then* $\rho_l = \rho_r = \rho$, *otherwise if* $n = |\rho|$ *then* $|\rho_l| < n$ *and* $|\rho_r| < n$.
3. *If* $\phi(\Sigma) = (\Sigma_l, \Sigma_r)$ *and* $\phi(\rho) = (\rho_l, \rho_r)$ *then* $\rho \models \Sigma$ *iff both* $\rho_l \models \Sigma_l$ *and* $\rho_r \models \Sigma_r$.
4. *If* $|\rho| = 0$ *then* $\rho_l = \rho_r = \rho$, *otherwise* $|\rho_l| < |\rho|$ *and* $|\rho_r| < |\rho|$. *If* $|\Sigma| = 0$ *then* $\Sigma_l = \Sigma_r = \Sigma$, *otherwise if* $|\Sigma_l| < |\Sigma|$ *and* $|\Sigma_r| < |\Sigma|$.

Proof. Proved in Coq. These results are generalized from Lemma **??**.

A.1 Domain reduction

Here we propose the *domain reduction* technique to reduce the search space, *e.g.* from infinite to finite.

Lemma 2. *We use uppercase letters to denote sets,* e.g., T *and* S. *Then:*

1. ***Emptiness***: *Let* $T \subseteq S$ *and* $f : S \mapsto T$. *Then* S *is empty iff* T *is empty Generally, let* $T_i \subseteq T_{i+1}$ *and* $f_i : T_{i+1} \mapsto T_i$ *for* $i = 1 \dots n$ *then*

$$\bigcup_{i=1}^{n+1} T_i \text{ is empty} \quad \text{iff} \quad T_1 \text{ is empty}$$

2. ***Inclusion***: *Let* $T \subseteq S$ *and* $f : S \mapsto T^k$ *a k-ary bijection s.t.* $f^{-1}((T \cap S')^k) \subseteq S'$ *then:*

$$S \subseteq S' \quad \text{iff} \quad T \subseteq S'$$

Generally, let $T_i \subseteq T_{i+1}$ *and* $f_i : T_{i+1} \mapsto T_i^{k_i}$ *be* k_i*-ary bijection for* $i = 1 \dots n$ *s.t.* $f_i^{-1}((T_i \cap S')^{k_i}) \subseteq S'$ *then*

$$\bigcup_{i=1}^{n+1} T_i \subseteq S' \quad \text{iff} \quad T_1 \subseteq S'$$

Proof. The emptiness problem is trivial so we only focus on the inclusion problem. Only the $\Leftarrow$ is nontrivial: assume $T \subseteq S'$ and let $x \in S$ then $f(x) \in f(S) = T^k$, Thus $f(x) \in T^k \cap S'^k = (T \cap S')^k$ which implies $x \in f^{-1}((T \cap S')^k) \subseteq S'$. This concludes $S \subseteq S'$. The general case is done by induction over n.

For **convenience**, we denote $S(\Sigma)$ to be the set of all solutions of Σ, and $S_i(\Sigma) \subseteq S(\Sigma)$ to be the set of all solutions of height at most i.

Proof (of ***SAT*** *and* ***IMP*** *in [?]).* To demonstrate the usefulness of our domain reduction technique, we will use it to simplify the proof of finite height for **SAT** and **IMP** in [**?**], *i.e.*, satisfiability and entailment of share systems without disequations.

Let $n = |\Sigma|$, we classify the solution space of Σ into $\{S_i(\Sigma)\}_{i=n}^{\infty}$ then $S_i(\Sigma) \subseteq S_{i+1}(\Sigma)$ and $\lfloor \overleftarrow{\rho} \rfloor_{i+1}$ is a mapping from $S_{i+1}(\Sigma)$ to $S_i(\Sigma)$. Thus by emptiness property in Lemma **??**, $\mathbf{SAT}(\Sigma)$ iff there exists a solution in $S_n(\Sigma)$.

For $\mathbf{IMP}(\Sigma, \Sigma')$, let $n = |(\Sigma, \Sigma')|$ be the height of the entailment $\Sigma \vdash \Sigma'$. Then for each $m > n$, the function $f_{m+1}(\rho) \overset{\text{def}}{=} \lfloor \rho \rceil_{m+1}$ is a bijection from $S_{m+1}(\Sigma)$ to $S_m^2(\Sigma')$. Here we override $\lfloor \cdot \rceil_{m+1}$ over contexts by applying the pair rounding component-wise, *i.e.*, $\lfloor \rho \rceil_{m+1} = (\rho_l, \rho_r)$ s.t.:

$$\rho_l(v) = \tau_1 \ \wedge \ \rho_r(v) = \tau_2 \quad \text{iff} \quad \lfloor \rho \rceil_{m+1}(v) = (\tau_1, \tau_2)$$

Furthermore, let ρ, ρ' be both solution of Σ and Σ' and their height is at most m, *i.e.*:

$$\rho, \rho' \in S_m(\Sigma) \cap S_m(\Sigma')$$

Then by Lemma **??**, the context $\rho'' \overset{\text{def}}{=} f_{m+1}^{-1}(\rho, \rho') = \rho \bigtriangledown_{m+1} \rho'$ is a solution of Σ' and its height is at most $m + 1$. Thus:

$$f_{m+1}^{-1}((S_m(\Sigma) \cap S(\Sigma'))^2) \ = \ f_{m+1}^{-1}((S_m(\Sigma) \cap S_m(\Sigma'))^2) \ \subseteq \ S(\Sigma')$$

By inclusion property in Lemma **??**, it is sufficient to consider only solutions of height at most n in $S_n(\Sigma)$.

A.2 Correctness proof of Theorem ??

We now proceed to verify the correctness of Algorithm **??** for SAT. The heart of SAT is the specialized solver SSAT for singleton systems. As a result, we need to verify that SSAT is sound. First, we provide several key insights about singleton systems, *e.g.*, how 'big' solution is decomposed into smaller ones:

Lemma 3. *Let Σ^η be a singleton system and ρ a context of Σ^η s.t. $|\rho| = n > |\Sigma^\eta|$ and $\lfloor \rho \rceil_n = (\rho_l, \rho_r)$ then:*

1. *If $\rho \models \Sigma^\eta$ then both $\rho_l \models \Sigma^+$ and $\rho_r \models \Sigma^+$, and either $\rho_l \models \Sigma^\eta$ or $\rho_r \models \Sigma^\eta$. Conversely, if one of ρ_l, ρ_r is a solution of Σ^+ and the other is a solution of Σ^η then the context $\rho = \rho_l \bigtriangledown_n \rho_r$ is a solution of Σ^η.*

2. *Let* $\phi(\rho) = (\rho_l, \rho_r)$ *and* $\phi(\Sigma^\eta) = (\Sigma_l, \Sigma_r)$*. If* $\rho \models \Sigma^\eta$ *then* $\rho_l \models \Sigma_l^+$ *and* $\rho_r \models \Sigma_r^+$*. Also, either* $\rho_l \models \Sigma_l$ *or* $\rho_r \models \Sigma_r$*.*
   Conversely, if either $\rho_l \models \Sigma_l^+ \ \wedge\ \rho_r \models \Sigma_r^\eta$ *or* $\rho_l \models \Sigma_l^\eta \ \wedge\ \rho_r \models \Sigma_r^+$ *then* $\rho \models \Sigma^\eta$*.*
3. *If* $\mathbf{SAT}(\Sigma^+) = \top$ *then:*

$$\mathbf{SSAT}(\Sigma^\eta) = \top \quad \textit{iff} \quad \mathbf{SSAT}(\Sigma_i) = \top \textit{ for some } \Sigma_i \in \mathsf{DECOMPOSE}(\Sigma^\eta).$$

4. *If* $|\Sigma^\eta| = 0$ *then* $\mathbf{SSAT}(\Sigma^\eta) = \top$ *iff it has a solution of height zero.*

Proof. Proved in Coq. Here we will briefly explain the proof idea. Prop. 1 and 2 can be derived directly from Lemma **??**.

For Prop. 3, it is sufficient to prove for ϕ instead of DECOMPOSE as the latter can be generalized by induction. For $\Rightarrow$, let $\rho \models \Sigma^\eta$, $\phi(\rho) = (\rho_l, \rho_r)$, $\phi(\Sigma^\eta) = (\Sigma_l^\eta, \Sigma_r^\eta)$. Then from Prop. 2, either $\rho_l \models \Sigma_l$ or $\rho_r \models \Sigma_r$. For $\Leftarrow$, w.l.o.g. assume $\rho_l \models \Sigma_l$. As $\mathsf{SAT}(\Sigma^+) = \top$, there exists $\rho' \models \Sigma^+$. Let $\phi(\rho') = (\rho_l', \rho_r')$ then $\phi^{-1}(\rho_l, \rho_r') \models \Sigma$.

For Prop. 4, notice if $|\Sigma^\eta| = 0$ then ϕ is the identity function,*i.e.*, $\phi(\Sigma^\eta) = (\Sigma^\eta, \Sigma^\eta)$. We conduct the proof using our domain reduction technique. First, we classify the solutions of Σ^η into $\{S_i(\Sigma^\eta)\}_{i=0}^\infty$. Then for each $i \in \mathbb{N}$, $S_i(\Sigma^\eta) \subseteq S_{i+1}(\Sigma^\eta)$ and if $\rho \in S_{i+1}(\Sigma^\eta)$ then either $\rho_l \in S_i(\Sigma^\eta)$ or $\rho_r \in S_i(\Sigma^\eta)$ by Prop. 3. As a result, we can define a mapping from $S_{i+1}(\Sigma^\eta)$ to $S_i(\Sigma^\eta)$. By the emptiness property in Lemma **??**, it suffices to search for solutions of height zero.

Using the above properties, we show that the specialized solver SSAT is sound with respect to appropriate pre-condition:

Lemma 4. *If* $\mathbf{SAT}(\Sigma^+) = \top$ *then* $\mathsf{SSAT}(\Sigma^\eta) = \top$ *iff* Σ^η *is satisfiable.*

Proof. By Prop. 3 in **??**, it is equivalent to the satisfiability of one of the singleton sub-systems of height zero in $\mathsf{DECOMPOSE}(\Sigma^\eta)$. By Prop 4 in **??**, such sub-system must have solution of height zero and thus can be transformed into a Boolean formula to be solved by SMT solver. Hence the result follows.

Our next step is to show that $\mathbf{SAT}(\Sigma)$ is equivalent to the conjunction of $\mathbf{SAT}(\Sigma^{\eta_i})$ for each disequation η_i in Σ. As a result, the main solver SAT can just simply call the specialized solver SSAT multiple times:

Lemma 5. *Let* Σ *be a share system then* $\boldsymbol{SAT}(\Sigma) = \top$ *iff* $\mathbf{SSAT}(\Sigma^{\eta_i})$ *for each disequation* η_i *in* Σ*.*

Proof. Let $[\eta_1, \ldots, \eta_n]$ be the disequation list in Σ and $A_i = S(\Sigma^{\eta_i})$ the solution space of each singleton system Σ^{η_i} then:

$$S(\Sigma) = \bigcap_{i=1}^{n} A_i$$

The original statement can be restated as "$S(\Sigma)$ is nonempty iff each A_i is nonempty". Only the $\Leftarrow$ is nontrivial as for the other direction, any solution in

$S(\Sigma)$ is also a solution in A_i. Let $\rho_i \in A_i$ be a solution of Σ^{η_i}. To construct a solution of Σ from $\{\rho_i\}_{i=1}^n$, we define a sequence of contexts $\{\rho'_i\}_{i=1}^n$ as follows:

$$\rho'_1 \overset{\text{def}}{=} \rho_1, \ \rho'_k \overset{\text{def}}{=} \rho'_{k-1} \bigtriangledown_{n_k} \rho_k$$

where $n_k = \max(|\rho'_{k-1}|, |\rho_k|, |\Sigma|) + 1$. Intuitively, the averaging function helps to 'accumulate' the disequations while preserves the satisfiability of the remaining equations and equalities, *i.e.*:

$$\rho'_j \ \in \ \bigcap_{i=1}^{j} A_i$$

As a result, $\rho'_n \ \in \ \bigcap_{i=1}^{n} A_i = S(\Sigma)$ is a solution of Σ.

Finally, we are ready to justify the correctness of Theorem **??**:

Theorem 3. *Let Σ be a share system then Σ is satisfiable iff* $\mathsf{SAT}(\Sigma) = \top$.

Proof. As $\mathbf{SAT}(\Sigma^+)$ is a necessary condition for $\mathbf{SAT}(\Sigma)$, if $\mathbf{SAT}(\Sigma^+) = \bot$ then we also have $\mathbf{SAT}(\Sigma) = \bot$. As a result, lines 2-4 are correct. We need this check to activate the condition for Lemma **??**.

By Lemma **??**, the system Σ is satisfiable iff $\mathbf{SSAT}(\Sigma^{\eta_i}) = \top$ for each disequation η_i in Σ. This justifies the conjunction in line 6. Last but not least, Lemma **??** provides the correctness for the specialized solver SSAT and thus completes the correctness proof for SAT as well.

A.3 Correctness proof of Theorem ??

Our decision procedure for entailments **IMP** makes use of two specialized solvers ZIMP and SIMP for Z-systems and S-systems respectively. Hence the correctness of IMP also relies on the correctness of ZIMP and SIMP. As a result, we first verify that our two specialized solvers are sound. First, we prove several essential properties about two entailment problems **ZIMP** and **SIMP**:

Lemma 6. *Let Σ_1, Σ_2 be share systems then:*

1. *If $\mathbf{SAT}(\Sigma_1^+) = \top$ and $\Sigma_1^+ \vdash \Sigma_2^+$ then:*

$$\Sigma_1^+ \vdash \Sigma_2^\eta \quad \textit{iff} \quad \Sigma_a \vdash \Sigma_b \textit{ for some } (\Sigma_a, \Sigma_b) \in \mathit{DECOMPOSE}(\Sigma_1^+, \Sigma_2^\eta)$$

2. *If $|(\Sigma_1^+, \Sigma_2^\eta)| = 0$ then:*

$$\Sigma_1^+ \vdash \Sigma_2^\eta \quad \textit{iff} \quad \Sigma_1^+ \vdash \Sigma_2^\eta \textit{ for all contexts of height zero}$$

3. *If $\mathbf{SAT}(\Sigma_1^+) = \top$ and $\Sigma_1^+ \vdash \Sigma_2^+$ and $\Sigma_1^+ \nvdash \Sigma_2^{\eta_2}$ then:*

$$\Sigma_1^{\eta_1} \vdash \Sigma_2^{\eta_2} \quad \textit{iff} \quad \Sigma_a \vdash \Sigma_b \textit{ for each } (\Sigma_a, \Sigma_b) \in \mathit{DECOMPOSE}(\Sigma_1^{\eta_1}, \Sigma_2^{\eta_2})$$

4. If $|(\Sigma_1^{\eta_1}, \Sigma_2^{\eta_2})| = 0$ *and* $\Sigma_1^+ \vdash \Sigma_2^+$ *then:*

$$\Sigma_1^{\eta_1} \vdash \Sigma_2^{\eta_2} \quad \textit{iff} \quad \Sigma_1^{\eta_1} \vdash \Sigma_2^{\eta_2} \textit{ for all contexts of height zero}$$

Proof. The preconditions here are essential for the soundness proof of our specialized solvers. Fortunately, all of them are also necessary conditions for **IMP** and can be checked by known decision procedures. To make things simple, we will prove over ϕ instead of DECOMPOSE as the latter can be generalized from the former.

1. For $\Rightarrow$, assume $\Sigma_{1l}^+ \nvdash \Sigma_{2l}^{\eta_l}$ and $\Sigma_{1r}^+ \nvdash \Sigma_{2r}^{\eta_r}$. Then we can find two contexts ρ_l and ρ_r s.t.:
$$\rho_l \models \Sigma_{1l}^+ \ \wedge\ \rho_l \not\models \Sigma_{2l}^{\eta_l} \quad \text{and} \quad \rho_r \models \Sigma_{1r}^+ \ \wedge\ \rho_r \not\models \Sigma_{2r}^{\eta_r} \tag{2}$$
Let $\rho = \phi^{-1}(\rho_l, \rho_r)$ then $\rho \models \Sigma_1^+$. As $\Sigma_1^+ \vdash \Sigma_2^{\eta}$, we derive that $\rho \models \Sigma_2^{\eta}$. Thus either $\rho_l \models \Sigma_{2l}^{\eta_l}$ or $\rho_r \models \Sigma_{2r}^{\eta_r}$. This is a contradiction to equation **??**.
For $\Leftarrow$, w.l.o.g. assume $\Sigma_{1l}^+ \vdash \Sigma_{2l}^{\eta_l}$. Let $\rho \models \Sigma_1^+$ then $\rho_l \models \Sigma_{1l}^+$ and thus $\rho_l \models \Sigma_{2l}^{\eta_l}$ by the entailment assumption. From the premise $\mathbf{SAT}(\Sigma_1^+) = \top$, we can find $\rho' \models \Sigma_1^+$. Combine with the entailment premise $\Sigma_1^+ \vdash \Sigma_2^+$, this gives us $\rho' \models \Sigma_2^+$. As a result, let $\phi(\rho') = (\rho'_l, \rho'_r)$ then $\rho'_r \models \Sigma_{2r}^+$. Finally, let $\rho'' \stackrel{\text{def}}{=} \phi^{-1}(\rho_l, \rho'_r)$. From the two results $\rho_l \models \Sigma_{1l}^{\eta_l}$ and $\rho'_r \models \Sigma_{2r}^+$, we conclude $\rho'' \models \Sigma_2^{\eta}$.
2. We prove using domain reduction. We classify the solution space $S(\Sigma_1^+)$ into $\{S_i(\Sigma_1^+)\}_{i=1}^{\infty}$ s.t. $S_i(\Sigma_1^+) \subseteq S_{i+1}(\Sigma_1^+)$ and let ϕ be the bijection from $S_{i+1}(\Sigma_1^+)$ to $S_i^2(\Sigma_1^+)$. Notice that ϕ is the identity function for systems of height zero, *i.e.*:
$$\phi(\Sigma_1^+, \Sigma_2^{\eta}) \ = \ ((\Sigma_1^+, \Sigma_2^{\eta}), (\Sigma_1^+, \Sigma_2^{\eta}))$$
Let ρ_l, ρ_r be two contexts of height at most k that both satisfy Σ_1^+ and Σ_2^{η}, *i.e.*:
$$\rho_l, \rho_r \ \in \ S_k(\Sigma_1^+) \cap S_k(\Sigma_2^{\eta})$$
Then it follows that the context $\rho \stackrel{\text{def}}{=} \phi^{-1}(\rho_l, \rho_r)$ is also a solution of Σ_2^{η}. By the inclusion property of domain reduction, it suffices to consider only height-0 solutions in $S_0(\Sigma_1^+)$.
3. For $\Rightarrow$, it suffices to prove $\Sigma_{1l}^{\eta_{1l}} \vdash \Sigma_{2l}^{\eta_{2l}}$. Let ρ_l be a context s.t. $\rho_l \models \Sigma_{1l}^{\eta_1}$, we will prove that $\rho_l \models \Sigma_{2l}^{\eta_{2l}}$. From two premises $\mathbf{SAT}(\Sigma_1^+)$ and $\mathbf{IMP}(\Sigma_1^+, \Sigma_2^+)$, we can find a context ρ' s.t.:
$$\rho' \models \Sigma_1^+ \quad \text{and} \quad \rho' \models \Sigma_2^+$$
Let $\phi(\rho') = (\rho'_l, \rho'_r)$ and $\rho \stackrel{\text{def}}{=} \phi(\rho_l, \rho'_r)$ then:
$$\rho'_r \models \Sigma_{1r}^+ \quad \text{and} \quad \rho_r \models \Sigma_{2r}^+ \quad \text{and} \quad \rho \models \Sigma_1^{\eta_1}$$

As a result, it follows from $\Sigma_1^{\eta_1} \vdash \Sigma_2^{\eta_2}$ that $\rho \models \Sigma_2^{\eta_2}$. Consequently, either $\rho_l \models \Sigma_{2l}^{\eta_{2l}}$ or $\rho'_r \models \Sigma_{2r}^{\eta_{2r}}$. On the other hand, the premise $\Sigma_1^+ \nvdash \Sigma_2^{\eta_2}$ gives us $\rho' \not\models \Sigma_2^{\eta_2}$ and thus $\rho'_r \not\models \Sigma_{2r}^{\eta_{2r}}$[2]. Hence it must be the case that $\rho_l \models \Sigma_{2l}^{\eta_{2l}}$.
For $\Leftarrow$, let $\rho \models \Sigma_1^{\eta_1}$ then either $\rho_l \models \Sigma_{1l}^{\eta_{1l}}$ or $\rho_r \models \Sigma_{1r}^{\eta_{1r}}$. W.l.o.g., assume $\rho_l \models \Sigma_{1l}^{\eta_{1l}}$ then the entailment $\Sigma_{1l}^{\eta_{1l}} \vdash \Sigma_{2l}^{\eta_{2l}}$ implies $\rho_l \models \Sigma_{2l}^{\eta_{2l}}$. From the premise $\Sigma_1^+ \vdash \Sigma_2^+$, we deduce $\rho \models \Sigma_2^+$. As a result, $\rho_r \models \Sigma_{2r}^+$. By combining two results $\rho_l \models \Sigma_{2l}^{\eta_{2l}}$ and $\rho_r \models \Sigma_{2r}^+$, we arrive at $\rho \models \Sigma_2^{\eta_2}$.

4. Only $\Leftarrow$ is nontrivial. It suffices to prove the case when solution has height 1 and the general case will follow by induction. As $|(\Sigma_1^{\eta_1}, \Sigma_2^{\eta_2})| = 0$, the ϕ returns the same share equation, *i.e.*:

$$\phi(\Sigma_1^{\eta_1}, \Sigma_2^{\eta_2}) \;=\; ((\Sigma_1^{\eta_1}, \Sigma_2^{\eta_2}), (\Sigma_1^{\eta_1}, \Sigma_2^{\eta_2}))$$

Let $\rho \in S_1(\Sigma_1^{\eta_1})$ be a height-1 solution of $\Sigma_1^{\eta_1}$ and $\phi(\rho) = (\rho_l, \rho_r)$ then either $\rho_l \in S_0(\Sigma_1^{\eta_1})$ or $\rho_r \in S_0(\Sigma_1^{\eta_1})$. W.l.o.g., assume $\rho_l \in S_0(\Sigma_1^{\eta_1})$ is a height-0 solution of $\Sigma_1^{\eta_2}$. Then by the premise, we deduce that $\rho_l \models \Sigma_2^{\eta_2}$.
On the other hand, the entailment $\Sigma_1^+ \vdash \Sigma_2^+$ gives us $\rho \models \Sigma_2^+$ and thus $\rho_r \models \Sigma_2^+$. By combining two results $\rho_l \models \Sigma_2^{\eta_2}$ and $\rho_r \models \Sigma_2^+$, we conclude that $\rho \models \Sigma_2^{\eta_2}$.

Using the above properties, we proceed to verify the correctness of ZIMP and SIMP (Algorithm **??**):

Lemma 7. *Let Σ_1, Σ_2 be share systems and η, η_1, η_2 disequations then:*

1. $\mathsf{ZIMP}(\Sigma_1^+, \Sigma_2^\eta) = \top$ *iff* $\Sigma_1^+ \vdash \Sigma_2^\eta$.
2. $\mathsf{SIMP}(\Sigma_1^{\eta_1}, \Sigma_2^{\eta_2}) = \top$ *iff* $\Sigma_1^{\eta_1} \vdash \Sigma_2^{\eta_2}$.

Proof. For entailment $\Sigma_1^+ \vdash \Sigma_2^\eta$, we activate the preconditions in Lemma **??** by first checking two necessary conditions $\mathbf{SAT}(\Sigma_1^+)$ and $\Sigma_1^+ \vdash \Sigma_2^+$. By Prop. 1 in Lemma **??**, it is sufficient to find a height-0 (Σ_a, Σ_b) in $\mathsf{DECOMPOSE}(\Sigma_1^+, \Sigma_2^\eta)$ s.t. $\Sigma_a \vdash \Sigma_b$. This justifies the disjunction form in line 4. Finally, Prop. 2 says that we only need to consider solutions of height zero and thus the entailment $\Sigma_a \vdash \Sigma_b$ can be transformed into Boolean formula and handled by SMT solver. This completes the soundness proof for ZIMP.

For entailment $\Sigma_1^{\eta_1} \vdash \Sigma_2^{\eta_2}$, we first check two necessary conditions $\mathbf{SAT}(\Sigma_1^+)$ and $\Sigma_1^+ \vdash \Sigma_2^+$ and the sufficient condition $\Sigma_1^+ \vdash \Sigma_2^{\eta_2}$. If no trivial result can be drawn from these conditions then by Prop. 3, it suffices to check $\Sigma_a \vdash \Sigma_b$ for each pair (Σ_a, Σ_b) in $\mathsf{DECOMPOSE}(\Sigma_1^{\eta_1}, \Sigma_2^{\eta_2})$. This justifies the conjunction form in line 11 of SIMP. By Prop. 4, each entailment $\Sigma_a \vdash \Sigma_b$ only needs to hold for solutions of height zero and thus they can be handled by SMT solver. Thus the correctness of SIMP is justified.

The second ingredient for soundness proof of **IMP** is the reduction from general entailment **GIMP** to the three special entailments **IMP**, **ZIMP** and **SIMP**:

[2] otherwise together with the fact $\rho_l \models \Sigma_{2l}^+$ we can deduce the contradiction $\phi^{-1}(\rho_l, \rho'_r) = \rho' \models \Sigma_2^{\eta_2}$

Lemma 8. *Let Σ_1, Σ_2 be share systems and l_1^-, l_2^- their disequation lists then:*

1. *If l_2^- is empty and $\boldsymbol{SAT}(\Sigma_1) = \top$ then:*

$$\Sigma_1 \vdash \Sigma_2 \quad \textit{iff} \quad \Sigma_1^+ \vdash \Sigma_2^+$$

2. *If $l_2^- = [\eta_1, \ldots, \eta_n]$ for $n > 0$ and l_1^- is empty then:*

$$\Sigma_1 \vdash \Sigma_2 \quad \textit{iff} \quad \Sigma_1 \vdash \Sigma_2^{\eta_i} \textit{ for } i = 1 \ldots n$$

3. *If $l_1^- = [\eta_1, \ldots, \eta_n]$ and $l_2^- = [\eta_1', \ldots, \eta_m']$ for $n, m > 0$ then:*

$$\Sigma_1 \vdash \Sigma_2 \quad \textit{iff} \quad \forall \eta_i' \in l_2^- .\ \exists \eta_j \in l_1^- .\ \Sigma_1^{\eta_j} \vdash \Sigma_2^{\eta_i'}$$

Proof. Similar to the proof of Lemma **??**.

1. Only $\Rightarrow$ is nontrivial. As l_2^- is empty, we have $\Sigma_2^+ = \Sigma_2$. Let $\rho \models \Sigma_1$, we will show that $\rho \models \Sigma_2$. As $\mathbf{SAT}(\Sigma_1) = \top$, we can find a context ρ' s.t. $\rho' \models \Sigma_1$. Let $n \stackrel{\text{def}}{=} \max(|\rho|, |\rho'|, |\Sigma_1|, |\Sigma_2|) + 1$, we combine two contexts ρ, ρ' into a single context ρ_n using the averaging function $\triangledown_n$, *i.e.*:

$$\rho_n \ = \ \rho \triangledown_n \rho'$$

   Then ρ_n is a solution of Σ_1. From the premise $\Sigma_1 \vdash \Sigma_2$, we derive that $\rho_n \models \Sigma_2 = \Sigma_2^+$. As $\lfloor \rho \rfloor_n = (\rho, \rho')$ and $n > |\Sigma_2|$, both ρ and ρ' are solutions of Σ_2 by Corollary **??**. Thus the result follows.
2. As $l_2^- = [\eta_1, \cdots, \eta_n]$, we have $S(\Sigma_2) \ = \ \bigcap_{i=1}^n S(\Sigma_2^{\eta_i})$. The entailment $\Sigma_1 \vdash \Sigma_2$ is equivalent to the set inclusion $S(\Sigma_1) \subseteq S(\Sigma_2)$. Similarly, $\Sigma_1 \vdash \Sigma_2^{\eta_i}$ is equivalent to $S(\Sigma_1) \subseteq S(\Sigma_2^{\eta_i})$ for $i = 1 \ldots n$. On the other hand, we have the following fact from basic set theory:

$$A \subseteq \bigcap_{i=1}^{n} B_i \quad \text{iff} \quad A \subseteq B_i \text{ for } i = 1 \ldots n$$

   Hence the result trivially follows from these observations.
3. Let $A_j \stackrel{\text{def}}{=} S(\Sigma_1^{\eta_j})$ for $j = 1 \ldots n$ and $B_i \stackrel{\text{def}}{=} S(\Sigma_2^{\eta_i'})$ for $i = 1 \ldots m$. The entailment $\Sigma_1 \vdash \Sigma_2$ is equivalent to:

$$\bigcap_{j=1}^{n} A_j \ \subseteq \ \bigcap_{i=1}^{m} B_i$$

   Also, the above set inclusion is equivalent to:

$$\bigcap_{j=1}^{n} A_j \ \subseteq \ B_i \text{ for } i = 1 \ldots m$$

   Fix a value for i. If there exists some A_j s.t. $A_j \subseteq B_i$ then we are done. This also means that $\Sigma_1^{\eta_j} \vdash \Sigma_2^{\eta_i'}$. Otherwise, assume $A_j \nsubseteq B_i$ for all $j = 1 \ldots n$, it

suffices to show that $\bigcap_{j=1}^{n} A_j \nsubseteq B_i$. Let $\rho_j \in A_j \backslash B_i$, *i.e.*, ρ_j is a solution of $A_j = S(\Sigma_1^{\eta_j})$ but it is not a solution of $B_i = S(\Sigma_2^{\eta_i'})$. We define the sequence $\{\rho_k'\}$ as follows:

$$\rho_1' \stackrel{\text{def}}{=} \rho_1, \ \rho_{k+1}' \stackrel{\text{def}}{=} \rho_k' \bigtriangledown_{h_{k+1}} \rho_{k+1}$$

where $h_{k+1} = \max(|\rho_k'|, |\rho_{k+1}|, |\Sigma_1|, |\Sigma_2|) + 1$. By Lemma **??**, we deduce that ρ_n' is a solution of each A_j for $j = 1 \ldots n$ and it is not a solution of B_i. As a result, we have $\rho_n' \in \bigcap_{j=1}^{n} A_j \backslash B_i$. Hence $\bigcap_{j=1}^{n} A_j \nsubseteq B_i$ and thus the result follows.

We are now ready to justify the soundness of IMP (Algorithm **??**) in Theorem **??**:

Theorem 4. *Let Σ_1, Σ_2 be share systems then $\Sigma_1 \vdash \Sigma_2$ iff* $\mathsf{IMP}(\Sigma_1, \Sigma_2) = \top$.

Proof. We first activate several necessary conditions $\mathsf{SAT}(\Sigma_1)$ (line 2) and $\Sigma_1^+ \vdash \Sigma_2^+$ (line 3). Then the subsequent flow of IMP follows from Lemma **??** which depends on the length of two disequation lists l_1^- and l_2^-. Here our main solver calls the two specialized solvers ZIMP and SIMP (line 7 and 10) whose soundness is verified in Lemma **??**. As a result, the soundness of IMP is justified.

B Development file list

Figure **??** contains a file-by-file summary of our development. Overall we have approximately 38.6k lines of code. These include roughly 5k lines of modifications to the Mechanized Semantic Library [**?**], which contained the original mechanized definitions of tree shares by Dockins *et al.* [**?**]. We have significantly expanded the theory to account *e.g.* for the operations of rounding and averaging explained in §**??** and formalized in §**??**. We did some of these MSL modifications to justify our previous decision procedure [**?**]. The rest of MSL adds about another 30k lines of code, but we do not include these lines in our table as we only use a small portion. The directory `/rbt/` contains a general and well-performing red-black tree implementation [**?**].

All of the other files are new for the present work. The `/uf/` directory uses the red-black trees to build our purely functional union-find implementation. The `/part/` directory uses union find to build our generic partition module.

The bulk of the files are in the main directory `/`. We define share equation system in `share_equation_system.v` and state the formal correctness property of our procedures in `share_dec_interface.v`. The INTERPRETER component is in `bool_to_formula.v` and `fbool_solver.v` contains the SMT solver; `bool_solver.v` chains these together.

The key theoretical proofs for domain reduction are in the files associated with §**??**. The code for the DECOMPOSER is in `share_decompose_base.v`, which defines the function ϕ, and `share_decompose.v`, which defines the DEC function. The code of TRANSFORMER is in `share_to_bool.v`; its correctness proof uses

all §**??** files. Lemmas **??** and **??** are proven in `share_correctness_base.v` and `share_correctness.v`.

The BOUNDER module is in `borders.v`, `bound_map.v`, and `share_bounder.v`. The PARTITION module is in `partition_modules.v`, which uses our generic partitioner. The SIMPLIFIER module is in `share_simplifier.v`.

The main procedures (SAT and IMP) and proofs are in `share_solver.v`. An optimized version in which we use the module PARTITION to divide the system into independent components are in `share_solver_with_partition.v`.

file	LOC	§
/msl/ (2 modified files)	≈ 5,000	3,A
boolean_alg.v	≈ 500	A
tree_shares.v	≈ 4,500	3,A
/rbt/ (3 files)	5,259	5
/uf/ (4 files)	3,881	5
base.v	268	5
UF_interface.v	935	5
UF_base.v	2,258	5
UF_implementation.v	420	5
/part/ (4 files)	2,729	5
base.v	268	5
partition_base.v	434	5
partition_ibase.v	1,094	5
partition_implementation.v	771	5
/ (20 files)	21,740	3–5
base.v	268	
share_dec_base.v	524	
base_properties	786	
share_equation_system.v	1,133	3
share_dec_interface.v	536	3
bool_to_formula.v	643	3
fbool_solver.v	1,497	3
bool_solver.v	417	3
share_correctness_base.v	1,594	4
share_correctness.v	447	4
share_decompose_base.v	2,265	4
share_decompose.v	670	4
share_to_bool.v	637	4
borders.v	464	5
bound_map.v	228	5
share_bounder.v	2,510	5
partition_modules.v	3,908	5
share_simplifier.v	2,337	5
share_solver.v	792	3
share_solver_with_partition.v	84	3
Total (31 files)	≈ 38,609	

Fig. 5. Our development