A New Decision Procedure for Finite Sets and Cardinality Constraints in SMT*

Kshitij Bansal[1], Andrew Reynolds[2], Clark Barrett[1], and Cesare Tinelli[2]

[1] Department of Computer Science, New York University
[2] Department of Computer Science, The University of Iowa

Abstract. We consider the problem of deciding the theory of finite sets with cardinality constraints using a satisfiability modulo theories solver. Sets are a common high-level data structure used in programming; thus, such a theory is useful for modeling program constructs directly. More importantly, sets are a basic construct of mathematics and thus natural to use when formalizing the properties of computational systems. We develop a calculus describing a modular combination of a procedure for reasoning about membership constraints with a procedure for reasoning about cardinality constraints. Cardinality reasoning involves tracking how different sets overlap. For efficiency, we avoid considering Venn regions directly, as done previous work. Instead, we develop a novel technique wherein potentially overlapping regions are considered incrementally as needed. We use a graph to track the interaction among the different regions. Initial experimental results demonstrate that the new technique is competitive with previous techniques and scales much better on certain classes of problems.

1 Introduction

Satisfiability modulo theories (SMT) solvers are at the heart of many verification tools. One of the reasons for their popularity is that fast, dedicated decision procedures for fragments of first-order logic are extremely useful for reasoning about constructs common in hardware and software verification. In particular, they provide a good balance between speed and expressiveness. Common fragments include theories such as bitvectors, arithmetic, and arrays, which are useful both for modeling basic constructs as well as for performing general reasoning.

As the use of SMT solvers has spread, there has been a corresponding demand for SMT solvers to support additional useful theories. Although it is possible to encode finitely axiomatizable theories using quantifiers, the performance and robustness gap between a custom decision procedure and an encoding using quantifiers can be quite significant.

In this paper, we present a new decision procedure for a fragment of set theory. Our main motivation is that sets are a common abstraction used in programming. As with other SMT theories like the theories of arrays and bitvectors,

* This work was partially supported by NSF grants 1228765, 1228768, and 1320583.

we expect the theory of sets to be useful in modeling a variety of program constructs. Sets are also used directly in high-level programming languages like SETL and in specification languages like Alloy, B and Z. More generally, sets are a basic construct in mathematics and come up quite naturally when trying to express properties of systems.

While the full language of set theory is undecidable, many interesting fragments are known to be decidable. We present a calculus which can handle basic set operations, such as membership, union, intersection, and difference, and which can also reason efficiently about set cardinalities. The calculus is also designed for easy integration into the DPLL(T) framework [12].

1.1 Related work

In the SMT community, the desire to support a theory of finite sets with cardinality goes back at least to a 2009 proposal [9]. However, the focus there is on formalizing the semantics and representation of the theory within the context of the SMT-LIB language, rather than on a decision procedure for deciding it.

There is an existing stream of research on exploring decidable fragments of set theory (often referred to in the literature as syllogistics) [5]. One such subfragment is MLSS, more precisely, the ground set-theoretic fragment with basic Boolean set operators (union, intersection, set difference), singleton operator and membership predicate. A tableau-based procedure for this fragment was presented in [6], and the part of our calculus covering this same fragment builds on that work. In [7], an extension of the theory of arrays is presented, which can be used to encode the MLSS fragment. However, this approach cannot be used to encode cardinality constraints.

In this paper, we consider the MLSS fragment extended with set cardinality operations. The decidability of this fragment was established in [14]. The procedure given there involves making an up-front guess that is exponential in the number of set variables, making it non-incremental and highly impractical. That said, the focus of [14] is on establishing decidability and not on providing an efficient procedure.

Another logical fragment that is closely related is the Boolean Algebra and Presburger Arithmetic (BAPA) fragment, for which several algorithms have been proposed [10,11,13]. Though BAPA doesn't have the membership predicate or the singleton operator in its language, [13, Section 4] shows how one can generalize the algorithm for such reasoning. Intuitively, singleton sets can be simulated by imposing a cardinality constraint $\mathsf{card}(X) = 1$. Similarly, a membership constraint, say $x \sqsubseteq S$, is encoded by introducing a singleton set, say X, and then using the subset operation: $X \sqsubseteq S$.

This reduction can lead to significant inefficiencies, however. Consider the following simple example: $x \sqsubseteq S_1 \sqcup (S_2 \sqcup (\ldots \sqcup (S_{99} \sqcup S_{100})))$. It is easy to see that the constraint is satisfiable. In our calculus, a straightforward repeated application of one of the rules for set unions can determine this. On the other hand, in a reduction to BAPA, the membership reasoning is reduced to reasoning about cardinalities of different sets. For example, the algorithm in [13] will reduce

the problem to arithmetic constraints involving variables for 2^{101} Venn regions derived from S_1, S_2, ..., S_{100}, and the singleton set introduced for x.

The broader point is that reasoning about cardinalities of Venn regions is the main bottleneck for this fragment. As we show in our calculus, it is possible to avoid using Venn regions for membership predicates by instead reasoning about them directly. For explicit cardinality constraints, our calculus minimizes the number of Venn regions that need to be considered by reasoning about only a limited number of relevant regions that are introduced lazily.

1.2 Formal Preliminaries

We work in the context of many-sorted first-order logic with equality. We assume the reader is familiar with the following notions: signature, term, literal, formula, free variable, interpretation, and satisfiability of a formula in an interpretation (see, e.g., [3] for more details). Let Σ be a many-sorted signature. We will use $\approx$ as the (infix) logical symbol for equality—which has type $\sigma \times \sigma$ for all sorts σ in Σ and is always interpreted as the identity relation. We write $s \not\approx t$ as an abbreviation of $\neg\, s \approx t$. If e is a term or a formula, we denote by $\mathcal{V}(e)$ the set of e's free variables, extending the notation to tuples and sets of terms or formulas as expected.

If φ is a Σ-formula and $\mathcal{I}$ a Σ-interpretation, we write $\mathcal{I} \models \varphi$ if $\mathcal{I}$ satisfies φ. If t is a term, we denote by $t^{\mathcal{I}}$ the value of t in $\mathcal{I}$. A *theory* is a pair $T = (\Sigma, \mathbf{I})$, where Σ is a signature and $\mathbf{I}$ is a class of Σ-interpretations that is closed under variable reassignment (i.e., every Σ-interpretation that differs from one in $\mathbf{I}$ only in how it interprets the variables is also in $\mathbf{I}$). $\mathbf{I}$ is also referred to as the *models* of T. A Σ-formula φ is *satisfiable* (resp., *unsatisfiable*) *in* T if it is satisfied by some (resp., no) interpretation in $\mathbf{I}$. A set Γ of Σ-formulas *entails in* T a Σ-formula φ, written $\Gamma \models_T \varphi$, if every interpretation in $\mathbf{I}$ that satisfies all formulas in Γ satisfies φ as well. We write $\models_T \varphi$ as an abbreviation for $\emptyset \models_T \varphi$. We write $\Gamma \models \varphi$ to denote that Γ entails φ in the class of all Σ-interpretations. The set Γ is *satisfiable in* T if $\Gamma \not\models_T \bot$ where $\bot$ is the universally false atom. Two Σ-formulas are *equisatisfiable in* T if for every model $\mathcal{A}$ of T that satisfies one, there is a model of T that satisfies the other and differs from $\mathcal{A}$ at most over the free variables not shared by the two formulas. When convenient, we will tacitly treat a finite set of formulas as the conjunction of its elements and vice versa.

2 A Theory of Finite Sets with Cardinality

We consider a typed theory $\mathfrak{T}_S$ of finite sets with cardinality. In a more general logical setting, this theory would be equipped with a parametric set type, with a type parameter for the set's elements, and a corresponding collection of polymorphic set operations.[3] For simplicity here, we will describe instead a many-sorted theory of sets of sort Set whose elements are all of sort Element.

[3] In fact, this is the setting supported in our implementation in CVC4.

Constant and function symbols:

n : Card for all $n \in \mathbb{N}$ | − : Card → Card | + : Card × Card → Card

$\emptyset$: Set | card$(\cdot)$: Set → Card | $\{\cdot\}$: Element → Set | $\sqcup, \sqcap, \setminus$: Set × Set → Set

Predicate symbols:

< : Card × Card | >= : Card × Card | $\sqsubseteq$: Set × Set | $\in$: Element × Set

Fig. 1: The signature of $\mathfrak{T}_S$.

The theory $\mathfrak{T}_S$ can be combined with any other theory $\mathfrak{T}$ in a standard way, i.e., Nelson-Oppen-style, by identifying the Element sort with a sort σ in $\mathfrak{T}$, with the restriction that σ must be interpreted in $\mathfrak{T}$ as an infinite set.[4] Note that we limit our language to consider only *flat* sets (i.e. no sets of sets). However, this can be simulated by combining $\mathfrak{T}$ with itself using the mechanism just mentioned. The theory $\mathfrak{T}_S$ has also a sort Card for terms denoting set cardinalities. Since we consider only finite sets, all cardinalities will be natural numbers.

Atomic formulas in $\mathfrak{T}_S$ are built over a signature with these three sorts, and an infinite set of variables for each sort. Modulo isomorphism, $\mathfrak{T}_S$ is the theory of a single many-sorted structure, and its models differ in essence only on how they interpret the variables. Each model of $\mathfrak{T}_S$ interprets Element as some *countably infinite* set E, Set as the set of *finite* subsets of E, and Card as $\mathbb{N}$. The signature of $\mathfrak{T}_S$ has the following predicate and function symbols, summarized in Figure 1: the usual symbols of *linear* integer arithmetic, the usual set composition operators, an empty set ($\emptyset$) and a singleton set ($\{\cdot\}$) constructor, and a cardinality operator (card$(\cdot)$), all interpreted as expected. The signature includes also symbols for the cardinality comparison (<), subset ($\sqsubseteq$) and membership ($\in$) predicates.

We call *set term* any term of sort Set or of the form card(s), and *cardinality term* any term of sort Card with no occurrences of card$(\cdot)$. A *set constraint* is an atomic formula of the form $s \approx t$, $s \sqsubseteq t$, $e \in t$ or their negation, with s and t set terms and e a term of sort Element. A *cardinality constraint* is a [dis]equality $[\neg]c \approx d$ or an inequality c < d or c >= d where c and d are cardinality terms. An *element constraint* is a [dis]equality $[\neg]x \approx y$ where x and y are variables of sort Element. A $\mathfrak{T}_S$*-constraint* is a set, cardinality or element constraint.

We will use x, y for variables of sort Element; S, T, U for variables of sort Set; s, t, u, v for terms of sort Set; and c with subscripts for variables of sort Card. Given $\mathcal{C}$, a set of constraints, Vars $(\mathcal{C})$ (respectively, Terms$(\mathcal{C})$) denotes the set of variables (respectively, terms) in $\mathcal{C}$. For notational convenience, we fix an injective mapping from terms of sort Set to variables of sort Card that allows us to associate to each such term s a unique cardinality variable c_s.

We are interested in checking the satisfiability in $\mathfrak{T}_S$ of finite sets of $\mathfrak{T}_S$-constraints. While this problem is decidable, it has high worst-case time complexity [14]. So our efforts are in the direction of producing a solver for $\mathfrak{T}_S$-constraints

[4] An extension that allows σ to be interpreted as finite by relying on polite combination [8] is planned as future work.

that is efficient in practice, in addition to being correct and terminating. Our solver relies on the modular combination of a solver for set constraints and an off-the-shelf solver for linear integer arithmetic, which handles arithmetic constraints over set cardinalities.

3 A Calculus for the Theory

In this section, we describe a tableaux-style calculus capturing the essence of our combined solver for $\mathfrak{T}_S$. As we describe in the next section, that calculus admits a proof procedure that decides the satisfiability of $\mathfrak{T}_S$-constraints.

For simplicity, we consider as input to the calculus only conjunctions $\mathcal{C}$ of constraints whose set constraints are in *flat form*. These are (well-sorted) set constraints of the form $S \approx T$, $S \not\approx T$, $S \approx \emptyset$, $S \approx \{x\}$, $S \approx T \sqcup U$, $S \approx T \sqcap U$, $S \approx T \setminus U$, $x \sqsubseteq S$, $x \not\sqsubseteq S$, or $c_S \approx \mathsf{card}(S)$, where S, T, U, c_S, and x are variables of the expected sort. We also assume that any set variable S of $\mathcal{C}$ appears in at most one union, intersection or set difference term. Thanks to common satisfiability-preserving transformations,[5] all of these assumptions can be made without loss of generality.

The calculus is described as a set of derivation rules which modify a *state* data structure. A state is either the special state unsat or a tuple of the form $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G} \rangle$, where $\mathcal{S}$ is a set of set constraints, $\mathcal{M}$ is a set of element constraints, $\mathcal{A}$ is a set of cardinality constraints, and $\mathcal{G}$ is a directed graph over set terms with nodes $V(\mathcal{G})$ and edges $E(\mathcal{G})$. Since cardinality constraints can be processed by a standard arithmetic solver, and element constraints by a simple equality solver,[6] we present and discuss only rules that deal with set constraints.

The derivation rules are provided in Figures 2 through 9 in *guarded assignment form*. In such form, the premises of a rule refer to the current state and the conclusion describes how each state component is changed, if at all, by the rule's application. A derivation rule *applies* to a state σ if all the conditions in the rule's premises hold for σ *and* the resulting state is different from σ. In the rules, we write S, t as an abbreviation for $S \cup \{t\}$. Rules with two or more conclusions separated by the symbol $\|$ are non-deterministic branching rules.

The rules are such that it is possible to generate a closed tableau (or *derivation tree*) from an initial state $\langle \mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0, \mathcal{G}_0 \rangle$, where $\mathcal{G}_0$ is an empty graph, if and only if the conjunction of all the constraints in $\mathcal{S}_0 \cup \mathcal{M}_0 \cup \mathcal{A}_0$ is unsatisfiable in $\mathfrak{T}_S$. Broadly speaking, the derivation rules can be divided into three categories. First are those that reason about membership constraints (of form $x \sqsubseteq S$). These rules only update the components $\mathcal{S}$ and $\mathcal{M}$ of the current state, although their premises may depend on other parts of the state, in particular, the nodes of the graph $\mathcal{G}$. Second are rules that handle constraints of the form $c_S \approx \mathsf{card}(S)$. The graph incrementally built by the calculus is central to satisfying these constraints. Third are rules for propagating element and cardinality constraints respectively to $\mathcal{M}$ and $\mathcal{A}$.

[5] Including replacing constraints of the form $s \sqsubseteq t$ with $s \approx (s \sqcap t)$.

[6] Recall that $\mathfrak{T}_S$ has no terms of sort $\mathsf{Element}$ besides variables.

$$\textsc{Union Down I}\quad \frac{x \not\sqsubseteq s \sqcup t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \lhd (x \not\sqsubseteq s) \lhd (x \not\sqsubseteq t)} \qquad \textsc{Union Down II}\quad \frac{x \sqsubseteq s \sqcup t \in \mathcal{S}^* \qquad \{u, v\} = \{s, t\} \qquad x \not\sqsubseteq u \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \lhd (x \sqsubseteq v)}$$

$$\textsc{Union Up I}\quad \frac{x \not\sqsubseteq s \in \mathcal{S}^* \qquad x \not\sqsubseteq t \in \mathcal{S}^* \qquad s \sqcup t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \lhd (x \not\sqsubseteq s \sqcup t)} \qquad \textsc{Union Up II}\quad \frac{x \sqsubseteq u \in \mathcal{S}^* \qquad u \in \{s, t\} \qquad s \sqcup t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \lhd (x \sqsubseteq s \sqcup t)}$$

$$\textsc{Inter Down I}\quad \frac{x \sqsubseteq s \sqcap t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \lhd (x \sqsubseteq s) \lhd (x \sqsubseteq t)} \qquad \textsc{Inter Down II}\quad \frac{x \not\sqsubseteq s \sqcap t \in \mathcal{S}^* \qquad \{u, v\} = \{s, t\} \qquad x \sqsubseteq u \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \lhd (x \not\sqsubseteq v)}$$

$$\textsc{Inter Up I}\quad \frac{x \sqsubseteq s \in \mathcal{S}^* \qquad x \sqsubseteq t \in \mathcal{S}^* \qquad s \sqcap t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \lhd (x \sqsubseteq s \sqcap t)} \qquad \textsc{Inter Up II}\quad \frac{x \not\sqsubseteq u \in \mathcal{S}^* \qquad u \in \{s, t\} \qquad s \sqcap t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \lhd (x \not\sqsubseteq s \sqcap t)}$$

$$\textsc{Union split}\quad \frac{x \sqsubseteq s \sqcup t \in \mathcal{S} \qquad x \sqsubseteq s, x \sqsubseteq t \notin \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \lhd (x \sqsubseteq s) \;\|\; \mathcal{S} := \mathcal{S} \lhd (x \sqsubseteq t)}$$

$$\textsc{Inter split}\quad \frac{s \sqcap t \in \mathcal{T} \qquad \{u, v\} = \{s, t\} \qquad x \sqsubseteq u \in \mathcal{S}^* \qquad x \sqsubseteq v, x \not\sqsubseteq v \notin \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \lhd (x \sqsubseteq v) \;\|\; \mathcal{S} := \mathcal{S} \lhd (x \not\sqsubseteq v)}$$

Fig. 2: Union and intersection rules.

3.1 Set reasoning rules

Figures 2 and 3 focus on sets without cardinality. They are based on the MLSS decision procedure by Cantone and Zarba [6], though with some key differences. First, the rules operate over a set $\mathcal{T}$ of Set terms which may be larger than just the terms in $\mathcal{S}$. This generalization is required because of additional terms that may be introduced when reasoning about cardinalities. Second, the reasoning is done modulo equality. A final, technical difference is that we work with sets of ur-elements rather than untyped sets.

These rules rely on the following additional notation. Given a set $\mathcal{C}$ of constraints, let $\text{Terms}_{\mathsf{Sort}}(\mathcal{C})$ refer to terms of sort Sort in $\mathcal{C}$, with $\text{Terms}(\mathcal{C})$ denoting all terms in $\mathcal{C}$. We define the binary relation $\approx^*_{\mathcal{C}} \subseteq \text{Terms}(\mathcal{C}) \times \text{Terms}(\mathcal{C})$ to be the reflexive, symmetric, and transitive closure of the relation on terms induced by equality constraints in $\mathcal{C}$. Now, we define the following closures:

$$\begin{aligned}
\mathcal{M}^* &= \{x \approx y \mid x \approx^*_{\mathcal{M}} y\} \cup \{x \not\approx y \mid \exists x', y'.\ x \approx^*_{\mathcal{M}} x',\ y \approx^*_{\mathcal{M}} y',\ x' \not\approx y' \in \mathcal{M}\} \\
\mathcal{S}^* &= \mathcal{S} \cup \{x \sqsubseteq s \mid \exists x', s'.\ x \approx^*_{\mathcal{M}} x',\ s \approx^*_{\mathcal{S}} s',\ x' \sqsubseteq s' \in \mathcal{S}\} \\
&\quad \cup \{x \not\sqsubseteq s \mid \exists x', s'.\ x \approx^*_{\mathcal{M}} x',\ s \approx^*_{\mathcal{S}} s',\ x' \not\sqsubseteq s' \in \mathcal{S}\}
\end{aligned}$$

where x, y, x', y' in $\text{Terms}_{\mathsf{Element}}(\mathcal{M} \cup \mathcal{S})$, and s, s' in $\text{Terms}_{\mathsf{Set}}(\mathcal{S})$. Next, we define a left-associative operator $\lhd$. Intuitively, given a set of constraints $\mathcal{C}$ and

$$\textsc{Singleton}\ \frac{\{x\} \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \lhd (x \sqsubseteq \{x\})} \qquad \textsc{Single Member}\ \frac{x \sqsubseteq \{y\} \in \mathcal{S}^*}{\mathcal{M} := \mathcal{M} \lhd (x \approx y)} \qquad \textsc{Single Non-member}\ \frac{x \not\sqsubseteq \{y\} \in \mathcal{S}^*}{\mathcal{M} := \mathcal{M} \lhd (x \not\approx y)}$$

$$\textsc{Set Disequality}\ \frac{s \not\approx t \in \mathcal{S}^* \qquad \nexists x \in \text{Terms}(\mathcal{S}) \text{ such that } x \sqsubseteq s \in \mathcal{S}^* \text{ and } x \not\sqsubseteq t \in \mathcal{S}^* \qquad \nexists x \in \text{Terms}(\mathcal{S}) \text{ such that } x \not\sqsubseteq s \in \mathcal{S}^* \text{ and } x \sqsubseteq t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \lhd (y \sqsubseteq s) \lhd (y \not\sqsubseteq t) \quad \| \quad \mathcal{S} := \mathcal{S} \lhd (y \not\sqsubseteq s) \lhd (y \sqsubseteq t)}$$

$$\textsc{Eq Unsat}\ \frac{(x \not\approx x) \in \mathcal{M}^*}{\mathsf{unsat}} \qquad \textsc{Set Unsat}\ \frac{(x \sqsubseteq s) \in \mathcal{S}^* \qquad (x \not\sqsubseteq s) \in \mathcal{S}^*}{\mathsf{unsat}} \qquad \textsc{Empty Unsat}\ \frac{(x \sqsubseteq \emptyset) \in \mathcal{S}^*}{\mathsf{unsat}}$$

Fig. 3: Singleton, disequality and contradiction rules. Here, y is a fresh variable.

a literal l, $\mathcal{C} \lhd (l)$ adds l to $\mathcal{C}$ only if l is not in $\mathcal{C}$'s closure. More precisely,

$$\mathcal{C} \lhd (l) = \begin{cases} \mathcal{C} & \text{if } l \in \mathcal{C}^* \\ \mathcal{C} \cup \{l\} & \text{otherwise.} \end{cases} \tag{1}$$

Finally, the set of *relevant* terms for these rules is denoted by $\mathcal{T}$ and consists of terms from $\mathcal{S}$ and $\mathcal{G}$: $\mathcal{T} = \text{Terms}(\mathcal{S}) \cup V(\mathcal{G})$.

Figure 2 shows the rules for reasoning about membership in unions and intersections. Each rule covers one case in which a new membership (or non-membership) constraint can be deduced. The justification for these rules is straightforward based on the semantics of the set operations. Due to space limitations, we do not show the rules that process set difference constraints. However, they are analogous to those given for union and intersection constraints. Figure 3 shows rules for singletons, disequalities, and contradictions. Note in particular that the Set Disequality rule introduces a variable y, denoting an element that is in one set but not in the other.

Example 1. Let $\mathcal{S} = \{S \approx A \sqcup B, S \approx C \sqcap D, x \sqsubseteq C, x \not\sqsubseteq D, y \not\sqsubseteq S, y \sqsubseteq D\}$. Using the rules in Figure 2, we can directly deduce the additional constraints: $x \not\sqsubseteq C \sqcap D$ (by Inter Up II), $x \not\sqsubseteq A$, $x \not\sqsubseteq B$, $y \not\sqsubseteq A$, $y \not\sqsubseteq B$ (by Union Down I), and $y \not\sqsubseteq C$ (by Inter Down II). This gives a complete picture, modulo equality, of exactly which sets contain x and y. □

3.2 Cardinality of sets

The next set of rules is based on two observations: (*i*) the cardinality of two sets, and that of their union, intersection and set difference are inter-related; (*ii*) if two set terms are asserted to be equal, their cardinalities must match. Figure 4 shows the Venn regions for two sets, T and U. Notice the following relationships: T is a disjoint union of $T \setminus U$ and $T \sqcap U$; $T \sqcup U$ is a disjoint union of $T \setminus U$ and $T \sqcap U$ and $U \setminus T$; and U is a disjoint union of $T \sqcap U$ and $U \setminus T$. Knowing that

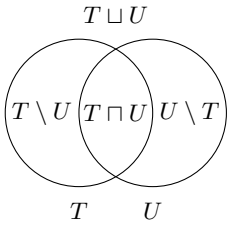

Fig. 4: Venn regions for T and U.

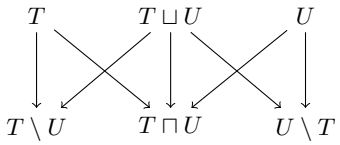

Fig. 5: The same structure as a graph.

the sets are *disjoint* is important; it allows us to infer the constraints:

$$\begin{aligned} \mathsf{card}(T) &\approx \mathsf{card}(T \setminus U) + \mathsf{card}(T \sqcap U) \\ \mathsf{card}(T \sqcup U) &\approx \mathsf{card}(T \setminus U) + \mathsf{card}(T \sqcap U) + \mathsf{card}(U \setminus T) \\ \mathsf{card}(U) &\approx \mathsf{card}(U \setminus T) + \mathsf{card}(T \sqcap U). \end{aligned}$$

We can represent these same relationships using a graph. The nodes of the graph are set terms, and each node has the property that it is the disjoint union of its children in the graph. The graph for the regions in Figure 4 is shown in Figure 5. We ensure that the graph contains all nodes whose cardinality is implicitly or explicitly constrained by the current state. Set terms with implicit cardinality constraints include (i) union, intersection, and set difference terms appearing in $\mathcal{S}$, for which one of the operands is already in the graph; and (ii) terms occurring in an equality whose other member is already in the graph. A careful analysis[7] reveals that we can actually avoid adding intersection terms $t \sqcap u$ unless both t and u are already in the graph, and set difference terms $t \setminus u$ unless t is already in the graph.

The rules in Figure 6 make use of a function add which takes a graph $\mathcal{G}$ and a term s and returns the graph $\mathcal{G}'$ defined as follows:

1. For $s = T$ or $s = \emptyset$ or $s = \{x\}$:
   $V(\mathcal{G}') = V(\mathcal{G}) \cup \{s\}$
   $E(\mathcal{G}') = E(\mathcal{G})$
2. For $s = T \sqcap U$ or $s = T \setminus U$:
   $V(\mathcal{G}') = V_2 = V(\mathcal{G}) \cup \{T, U, T \setminus U, T \sqcap U, U \setminus T\}$
   $E(\mathcal{G}') = E_e = E(\mathcal{G}) \cup \{(T, T \setminus U), (T, T \sqcap U), (U, T \sqcap U), (U, U \setminus T)\}$
3. For $s = T \sqcup U$ and V_2 and E_2 as above:
   $V(\mathcal{G}') = V_2 \cup \{T \sqcup U\}$
   $E(\mathcal{G}') = E_2 \cup \{(T \sqcup U, T \setminus U), (T \sqcup U, T \sqcap U), (T \sqcup U, U \setminus T)\}$

Recall that, by assumption, each set variable participates in at most one union, intersection, or set difference. This ensures that edges from a set variable node are added only once, maintaining the invariant that its children in the graph are disjoint. Terms with explicit constraints on their cardinality are added to

[7] See completeness proof in [1, Chapter 2] for further details.

$$\textsc{Introduce Eq Right}\quad \frac{S \approx t \in \mathcal{S} \qquad S \in V(\mathcal{G}) \qquad t \notin V(\mathcal{G})}{\mathcal{G} := \mathrm{add}(\mathcal{G}, t)}$$

$$\textsc{Introduce Union}\quad \frac{S \approx T \sqcup U \in \mathcal{S} \qquad T \sqcup U \notin V(\mathcal{G}) \qquad T \in V(\mathcal{G}) \text{ or } U \in V(\mathcal{G})}{\mathcal{G} := \mathrm{add}(\mathcal{G}, T \sqcup U)}$$

$$\textsc{Introduce Eq Left}\quad \frac{S \approx t \in \mathcal{S} \qquad S \notin V(\mathcal{G}) \qquad t \in V(\mathcal{G})}{\mathcal{G} := \mathrm{add}(\mathcal{G}, S)}$$

$$\textsc{Introduce Inter}\quad \frac{S \approx T \sqcap U \in \mathcal{S} \qquad T \sqcap U \notin V(\mathcal{G}) \qquad T \in V(\mathcal{G}) \qquad U \in V(\mathcal{G})}{\mathcal{G} := \mathrm{add}(\mathcal{G}, T \sqcap U)}$$

$$\textsc{Introduce Card}\quad \frac{c_s \approx \mathsf{card}(S) \in \mathcal{S}}{\mathcal{G} := \mathrm{add}(\mathcal{G}, S)} \qquad \textsc{Introduce Singleton}\quad \frac{\{x\} \in \mathrm{Terms}(\mathcal{S})}{\mathcal{G} := \mathrm{add}(\mathcal{G}, \{x\})} \qquad \textsc{Introduce Empty Set}\quad \frac{}{\mathcal{G} := \mathrm{add}(\mathcal{G}, \emptyset)}$$

Fig. 6: Graph introduction rules.

$$\textsc{Merge Equality I}\quad \frac{s \approx t \in \mathcal{S} \qquad s, t, \emptyset \in V(\mathcal{G}) \qquad \{u, v\} = \{s, t\} \qquad \mathcal{L}(u) \subsetneq \mathcal{L}(v)}{\mathcal{S} := \left\{ s' \approx \emptyset \;\middle|\; s' \in \mathcal{L}(v) \setminus \mathcal{L}(u) \right\} \cup \mathcal{S}}$$

$$\textsc{Merge Equality II}\quad \frac{s \approx t \in \mathcal{S} \qquad s, t \in V(\mathcal{G}) \qquad \mathcal{L}(s) \not\subseteq \mathcal{L}(t) \qquad \mathcal{L}(t) \not\subseteq \mathcal{L}(s)}{\mathcal{G} := \mathrm{merge}(\mathcal{G}, s, t)}$$

Fig. 7: Merge rules.

the graph by Introduce Card. Terms that have implicit constraints on their cardinality, specifically, singletons and the empty set, are added by rules Introduce Singleton and Introduce Empty Set.

If two nodes s and t in the graph are asserted to be equal (that is, $s \approx t \in \mathcal{S}$ or $t \approx s \in \mathcal{S}$), we can ensure they have the same cardinality by systematically modifying the graph. Let $\mathcal{L}(n)$ denote the set of leaf nodes for the subtree rooted at node n which are not known to be empty. Formally,

$$\mathcal{L}(n) = \{ n' \in \mathrm{Leaves}\,(n) \mid n' \approx \emptyset \notin \mathcal{S}^* \}\,, \tag{2}$$

where $\mathrm{Leaves}\,(v) = \{ w \in V(\mathcal{G}) \mid C(w) = \emptyset, w \text{ is reachable from } v \}$ and $C(w)$ denotes the children of w. We call two nodes n and n' *merged* if they have the same set of nonempty leaves, that is if $\mathcal{L}(n) = \mathcal{L}(n')$.

The rules in Figure 7 ensure that for all equalities over set terms, the corresponding nodes in the graph are merged. Consider an equality $s \approx t$. Rule Merge Equality I handles the case when either $\mathcal{L}(s)$ or $\mathcal{L}(t)$ is a proper subset of the other by constraining the extra leaves in the superset to be empty. Rule Merge Equality II handles the remaining case where neither is a subset of the other. The graph $\mathcal{G}' = \mathrm{merge}(\mathcal{G}, s, t)$ is defined as follows, where $L_1 = \mathcal{L}(s) \setminus \mathcal{L}(t)$ and $L_2 = \mathcal{L}(t) \setminus \mathcal{L}(s)$:

$$\begin{aligned} V(\mathcal{G}') &= V(\mathcal{G}) \cup \{ l_1 \sqcap l_2 \mid l_1 \in L_1, l_2 \in L_2 \} \\ E(\mathcal{G}') &= E(\mathcal{G}) \cup \{ (l_1, l_1 \sqcap l_2), (l_2, l_1 \sqcap l_2) \mid l_1 \in L_1, l_2 \in L_2 \} \end{aligned}$$

$$\textsc{Arithmetic contradiction}\quad \frac{\mathcal{A} \cup \hat{\mathcal{G}} \models_{\mathfrak{T}_A} \bot}{\mathsf{unsat}} \qquad \textsc{Guess Empty Set}\quad \frac{t \in \text{Leaves}\,(\mathcal{G})}{\mathcal{S} := \mathcal{S} \lhd (t \approx \emptyset) \quad \| \quad \mathcal{S} := \mathcal{S} \lhd (t \not\approx \emptyset)}$$

Fig. 8: Additional graph rules.

$$\textsc{Members Arrangement}\quad \frac{t \in \text{Leaves}\,(\mathcal{G}) \quad \mathcal{A} \not\Rrightarrow c_t \geq |t_{\mathcal{S}}| \quad [x]_{\mathcal{E}}, [y]_{\mathcal{E}} \in t_{\mathcal{S}} \quad [x]_{\mathcal{E}} \neq [y]_{\mathcal{E}} \quad x \not\approx y \notin \mathcal{M}^*}{\mathcal{M} := \mathcal{M} \lhd (x \approx y) \quad \| \quad \mathcal{M} := \mathcal{M} \lhd (x \not\approx y)}$$

$$\textsc{Guess Lower Bound}\quad \frac{t \in \text{Leaves}\,(\mathcal{G}) \qquad \mathcal{A} \not\Rrightarrow c_t \geq |t_{\mathcal{S}}| \qquad c_t < |t_{\mathcal{S}}| \notin \mathcal{A}}{\mathcal{A} := c_t \texttt{>=} |t_{\mathcal{S}}|, \mathcal{A} \quad \| \quad \mathcal{A} := c_t \texttt{<} |t_{\mathcal{S}}|, \mathcal{A}}$$

$$\textsc{Propagate Minsize}\quad \frac{x_1 \sqsubseteq s, \ldots, x_n \sqsubseteq s \in \mathcal{S}^* \quad \mathcal{A} \not\Rrightarrow c_s \geq n \qquad x_i \not\approx x_j \in \mathcal{M}^* \text{ for all } 1 \leq i < j \leq n}{\mathcal{A} := c_s \texttt{>=} n, \mathcal{A}}$$

Fig. 9: Cardinality and membership interaction rules.

We denote by $\hat{\mathcal{G}}$ the collection of all of the following arithmetic constraints imposed by graph $\mathcal{G}$:

1. For each set term $s \in V(\mathcal{G})$, its corresponding cardinality is the sum of the corresponding non-empty leaf nodes: $\left\{ c_s \approx \sum_{t \in \mathcal{L}(s)} c_t \;\middle|\; s \in V(\mathcal{G}) \right\}$.
2. Each cardinality is non-negative: $\{c_s \texttt{>=} 0 \mid s \in V(\mathcal{G})\}$.
3. A singleton set has cardinality 1: $\{c_s \approx 1 \mid s \in V(\mathcal{G}),\ s = \{x\}\}$.
4. The empty set has cardinality 0: $\{c_s \approx 0 \mid s \in V(\mathcal{G}),\ s = \emptyset\}$.

Rule Arithmetic contradiction, shown in Figure 8 makes use of the arithmetic solver to check whether the constraints in $\hat{\mathcal{G}}$ are inconsistent with the input constraints. Also shown is rule Guess Empty Set which can be used to guess if a leaf node is empty. This is useful to apply early on, to reduce the impact of merge operations on the size of the graph. Here and in Figure 9, $\text{Leaves}\,(\mathcal{G}) = \{v \in V(\mathcal{G}) \mid C(v) = \emptyset\}$.

3.3 Cardinality and membership interaction

The rules in Figure 9 propagate consequences of set membership constraints to the sets $\mathcal{M}$ and $\mathcal{A}$. Let $\mathcal{E}$ denote the set of equalities in $\mathcal{M}$, and let $[x]_{\mathcal{E}}$ denote the equivalence class of x with respect to $\mathcal{E}$. Then for a Set term t, $t_{\mathcal{S}} = \{[x]_{\mathcal{E}} \mid x \sqsubseteq t \in \mathcal{S}^*\}$, the set of equivalence classes of elements known to be in t. The notation $\mathcal{A} \Rrightarrow c_t \geq n$ means that $c_t \texttt{>=} k \in \mathcal{A}$ for some concrete constant $k \geq n$.

Rule Members Arrangement is used to decide which elements of a set should be equal or disequal. Once applied to completion, Rule Propagate Minsize can then be used to determine a lower bound for the cardinality of that set. Rule Guess Lower Bound can be used to short-circuit this process by guessing a conservative lower bound based on the number of distinct equivalence classes of elements known to be members of a set. If this does not lead

to a contradiction, a model can be found without resorting to extensive use of MEMBERS ARRANGEMENT.

Example 2. Consider again the constraints from Example 1, but now augmented with cardinality constraints $\{c_S \approx \mathsf{card}(S), c_C \approx \mathsf{card}(C), c_D \approx \mathsf{card}(D)\}$ and arithmetic constraints $\{c_S \texttt{ >= } 4, c_C + c_D \texttt{ < } 10\}$. Using the rules in Figure 6, the following nodes get added to the graph: S, C, D (by INTRODUCE CARD), $A \sqcup B$, $C \sqcap D$ (by INTRODUCE EQ RIGHT). $A \sqcup B$ is added with children $A \setminus B$, $A \sqcap B$, and $B \setminus A$; and by adding $C \sqcap D$, we also get $C \setminus D$ and $D \setminus C$, with the corresponding edges from C and D. Now, using two applications of MERGE EQUALITY II, we force the sets S, $A \sqcup B$ and $C \sqcap D$ to have the same set of 3 leaves, labeled $S \sqcap (A \setminus B) \sqcap (C \sqcap D)$, $S \sqcap (A \sqcap B) \sqcap (C \sqcap D)$, and $S \sqcap (B \setminus A) \sqcap (C \sqcap D)$. Let us call these nodes l_1, l_2, and l_3 for convenience. Let us also designate $l_4 = C \setminus D$ and $l_5 = D \setminus C$. Notice that the induced arithmetic constraints now include $c_S \approx c_{l_1} \texttt{ + } c_{l_2} \texttt{ + } c_{l_3}$, $c_C \approx c_{l_1} \texttt{ + } c_{l_2} \texttt{ + } c_{l_3} \texttt{ + } c_{l_4}$, and $c_D \approx c_{l_1} \texttt{ + } c_{l_2} \texttt{ + } c_{l_3} \texttt{ + } c_{l_5}$. With the addition of $C \setminus D$ and $D \setminus C$ to the graph, these are also added to $\mathcal{T}$. We can then deduce $x \sqsubseteq C \setminus D$ and $y \sqsubseteq D \setminus C$ using the (not shown) rules for propagation over set difference. Finally, we can use PROPAGATE MINSIZE to deduce $c_{l_4} \texttt{ >= } 1$ and $c_{l_5} \texttt{ >= } 1$. It is now not hard to see that using pure arithmetic reasoning, we can deduce that $c_C \texttt{ + } c_D \texttt{ >= } 10$ which leads to unsat using ARITHMETIC CONTRADICTION. □

4 Calculus Correctness

Our calculus is terminating and sound for any derivation strategy, that is, regardless of how the rules are applied. It is also refutation complete for any *fair* strategy, defined as a strategy that does not delay indefinitely the application of an applicable derivation rule. For space reasons, we only outline the proof arguments here. Complete proofs are given in [1].

We group the derivation rules of the calculus in the following subsets.

$\mathcal{R}_1$**:** membership predicate reasoning rules, from Figures 2 and 3.
$\mathcal{R}_2$**:** graph rules to reason about cardinality, from Figures 6, 7 and 8.
$\mathcal{R}_3$**:** rules from Figure 9 other than Rule GUESS LOWER BOUND.
$\mathcal{R}_4$**:** Rule GUESS LOWER BOUND.

The rules are used to construct derivation trees. A *derivation tree* is a tree over states, where the root is a state of the form $\langle \mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0, (\emptyset, \emptyset) \rangle$, (and $\mathcal{S}_0$, $\mathcal{M}_0$, $\mathcal{A}_0$ obey the input constraints mentioned at the beginning of Sec. 3), and where the children of each non-root node are obtained by applying one of the derivation rules of the calculus to that node. A branch of a derivation tree is *closed* if it ends with unsat; it is *saturated* with respect to a set $\mathcal{R}$ of rules if it is not closed and no rules in $\mathcal{R}$ apply to its leaf. A derivation tree is *closed* if all of its branches are closed. A derivation tree *derives* from a derivation tree T if it is obtained from T by the application of exactly one of the derivation rules to one of T's leaves.

Let S be a set of $\mathfrak{T}_S$-constraints. A *derivation (of S)* is a sequence $(T_i)_{0 \leq i \leq \kappa}$ of derivation trees, with κ finite or countably infinite, such that T_{i+1} derives from

T_i for all i, and T_0 is a one-node tree whose root is a state $\langle \mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0, (\emptyset, \emptyset)\rangle$ where $\mathcal{S}_0 \cup \mathcal{M}_0 \cup \mathcal{A}_0$ is $\mathfrak{T}_S$-equisatisfiable with S. A *refutation (of S)* is a (finite) derivation of S that ends with a closed tree.

4.1 Termination

Proposition 1 (Termination). *Every derivation in the calculus is finite.*

Proof (Sketch). It is enough to show that every application of a derivation rule to a state produces smaller states with respect to a well-founded relation $\succ$ over states other than unsat. For simplicity, we ignore the rule Guess Lower Bound, although the proof could be extended to that rule as well. To define $\succ$ we first define the following functions, each of which maps a state $\sigma = \langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G}\rangle$ to a natural number (from $\mathbb{N}$).

- $f_1(\sigma)$: number of equalities $t_1 \approx t_2$ in $\mathcal{S}$ such that either $t_1 \notin V(\mathcal{G})$, $t_2 \notin V(\mathcal{G})$, or $\mathcal{L}(t_1) \neq \mathcal{L}(t_2)$.
- $f_2(\sigma)$: cardinality of $(\mathrm{Terms}_{\mathsf{Set}}(\mathcal{S}) \cup \{\emptyset\}) \setminus V(\mathcal{G})$.
- $f_3(\sigma)$: cardinality of $\{t \in \mathrm{Leaves}(\mathcal{G}) \mid t \approx \emptyset \notin \mathcal{S}^*, t \not\approx \emptyset \notin \mathcal{S}^*\}$.
- $f_4(\sigma)$: number of disequalities $t_1 \not\approx t_2$ in $\mathcal{S}$ such that the premise of Set Disequality holds.
- $f_5(\sigma)$: cardinality of $T = \mathrm{Terms}_{\mathsf{Set}}(\mathcal{S}) \cup \{\emptyset\} \cup V(\mathcal{G})$.
- $f_6(\sigma)$: cardinality of $\mathrm{Terms}_{\mathsf{Element}}(\mathcal{S} \cup \mathcal{M})$.
- $f_7(\sigma)$: $2 \cdot f_6(\sigma)^2$ minus the cardinality of $\mathcal{M}^*$.[8]
- $f_8(\sigma)$: $2 \cdot f_5(\sigma)^2 + 2 \cdot f_5(\sigma) \cdot f_6(\sigma)$ minus the cardinality of $\mathcal{S}^*$.[9]
- $f_9(\sigma)$: cardinality of $T \setminus \{t \in \mathrm{Leaves}(\mathcal{G}) \mid \mathcal{A} \not\Rightarrow c_t \geq |t_{\mathcal{S}}|\}$.

Let $\left(\mathbb{N}^9, >^9_{\mathsf{lex}}\right)$ be the 9-fold lexicographic product of $(\mathbb{N}, >)$. We define $\succ$ as the relation such that $\sigma \succ \sigma'$ iff $(f_1(\sigma), \ldots, f_9(\sigma)) >^9_{\mathsf{lex}} (f_1(\sigma'), \ldots, f_9(\sigma'))$. □

4.2 Completeness

We develop the proof in stages, proving properties about different subsets of rules. We start with a proposition about the rule set $\mathcal{R}_1$.

Proposition 2. *Let $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G}\rangle$ be a state to which none of rules in $\mathcal{R}_1$ apply. There is a model $\mathfrak{S}$ of $\mathfrak{T}_S$ that satisfies the constraints $\mathcal{S}$ and $\mathcal{M}$ and has the following properties.*

1. *For all $x, y \in \mathrm{Vars}(\mathcal{M}) \cup \mathrm{Vars}(\mathcal{S})$ of sort Element, $x^{\mathfrak{S}} = y^{\mathfrak{S}}$ if and only if $x \approx y \in \mathcal{M}^*$.*
2. *For all $S \in \mathrm{Vars}(\mathcal{S})$ of sort Set, $S^{\mathfrak{S}} = \left\{x^{\mathfrak{S}} \;\middle|\; x \sqsubseteq S \in \mathcal{S}^*\right\}$.*
3. *For all $c_S \in \mathrm{Vars}(\mathcal{S})$ of sort Card, $c_S^{\mathfrak{S}} = \left|S^{\mathfrak{S}}\right|$.*

[8] Note that the cardinality of $\mathcal{M}^*$ is at most $2 \cdot (f_6(\sigma))^2$.

[9] One can show that this value is non-negative.

For the next two results, let $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G} \rangle$ be the leaf of a branch saturated with respect to rules $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$ in a derivation tree. The first result is about the effects of the rules in $\mathcal{R}_2$. The second is about the rules in $\mathcal{R}_3$.

Proposition 3. *For every $s \in V(\mathcal{G})$ the following holds.*

1. *If $s \approx t \in \mathcal{S}$ or $t \approx s \in \mathcal{S}$ for some t, then $\mathcal{L}(s) = \mathcal{L}(t)$.*
2. *If $s = T \sqcup U$, then $\mathcal{L}(T \sqcup U) = \mathcal{L}(T) \cup \mathcal{L}(U)$.*
3. *If $s = T \sqcap U$, then $\mathcal{L}(T \sqcap U) = \mathcal{L}(T) \cap \mathcal{L}(U)$.*
4. *If $s = T \setminus U$, then $\mathcal{L}(T \setminus U) = \mathcal{L}(T) \setminus \mathcal{L}(U)$.*
5. *For all distinct $t, u \in \text{Leaves}(s)$, $\models_{\mathfrak{T}_S} t \sqcap u \approx \emptyset$.*
6. *$\{t \approx u \mid t \approx u \in \mathcal{S}^*\} \models_{\mathfrak{T}_S} s \approx \bigsqcup_{t \in \mathcal{L}(s)} t$.*[10]

Proposition 4. *Let $\mathfrak{S}$ be an interpretation as the one specified in Proposition 2 and let $\mathfrak{A}$ be any model of $\mathfrak{T}_S$ satisfying $\mathcal{A}$. Then, for all $t \in \mathcal{L}(\mathcal{G})$, $c_t^{\mathfrak{A}} \geq \left| t^{\mathfrak{S}} \right|$.*

Completeness is a direct consequence of the following result.

Proposition 5. *Let $\mathcal{D}$ be a derivation tree with root $\langle \mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0, (\emptyset, \emptyset) \rangle$. If $\mathcal{D}$ has a branch saturated with respect to rules $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$, then there exists a model $\mathfrak{I}$ of $\mathfrak{T}_S$ that satisfies $\mathcal{S}_0 \cup \mathcal{M}_0 \cup \mathcal{A}_0$.*

Proof (Sketch). We build the model of the leaf nodes in the graph by modifying as needed the model obtained from Proposition 2. We add additional elements to these sets to make the cardinalities match the model satisfying the arithmetic constraints and the constraints induced by the graph. Propositions 3 and 4 ensure that it is always possible to do so without violating the set constraints. □

Proposition 6 (Completeness). *Under any fair derivation strategy, every derivation of a set S of $\mathfrak{T}_S$-unsatisfiable constraints extends to a refutation.*

Proof. Contrapositively, suppose that S has a derivation $\mathbf{D}$ that cannot be extended to a refutation. By Proposition 1, $\mathbf{D}$ must be extensible to one that ends with a tree with a saturated branch. By Proposition 5, S is satisfiable in $\mathfrak{T}_S$. □

4.3 Soundness

We start by showing that every rule preserves constraint satisfiability.

Lemma 1. *For every rule of the calculus, the premise state is satisfied by a model $\mathfrak{I}_p$ of $\mathfrak{T}_S$ iff one of its conclusion configurations is satisfied by a model $\mathfrak{I}_c$ of $\mathfrak{T}_S$ where $\mathfrak{I}_p$ and $\mathfrak{I}_c$ agree on the variables shared by the two states.*

[10] Technically, $\bigsqcup_{\dots}$ is ambiguous. But, for any structure in $\mathfrak{T}_S$, the interpretation of $\sqcup$ is associative, so the bracketing doesn't matter in our context.

Proof (Sketch). Soundness of the rules in Figure 2 and Figure 3 follows trivially from the semantics of set operators and the definition of $\mathcal{S}^*$. Soundness of MERGE EQUALITY I follows from properties of the graph (see Proposition 3, in particular the property that leaf terms are disjoint). The rules in Figure 6 and rule MERGE EQUALITY II do not modify the constraints, but we need them to establish properties of the graph. Soundness of the induced graph constraints in ARITHMETIC CONTRADICTION follows from Proposition 3 (in particular properties 5 and 6). Soundness of PROPAGATE MINSIZE follows from the semantics of cardinality. Soundness of GUESS EMPTY SET, MEMBERS ARRANGEMENT and GUESS LOWER BOUND is trivial. □

Proposition 7 (Soundness). *Every set of $\mathfrak{T}_S$-constraints that has a refutation is $\mathfrak{T}_S$-unsatisfiable.*

Proof (Sketch). Given Lemma 1, one can show by structural induction on derivation trees that the root of any closed derivation tree is $\mathfrak{T}_S$-unsatisfiable. The claim then follows from the fact that every refutation of a set S of $\mathfrak{T}_S$-constraints starts with a state $\mathfrak{T}_S$-equisatisfiable with S. □

5 Evaluation

We have implemented a decision procedure based on the calculus above in the SMT solver CVC4 [2]. We describe a high-level, non-deterministic version of it here, followed by an initial evaluation on benchmarks from program analysis.

5.1 Derivation strategy

The decision procedure can be thought of as a specific strategy for applying the rules given in Section 3, divided into the sets $\mathcal{R}_1, \ldots, \mathcal{R}_4$ introduced in Section 4.

Our derivation strategy can be summarized as follows. We start with the derivation from the initial state $\langle \mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0, \mathcal{G}_0 \rangle$ with $\mathcal{G}_0$ the empty graph, as described in Section 3, and apply the steps listed below, in the given order. The steps are described as rules being applied to a *current* branch. Initially, the current branch is the only branch in tree. On application of a rule with more than one conclusion, we select one of the branches (say, the left branch) as the current branch.

1. If a rule that derives `unsat` is applicable to the current branch, we apply one and close the branch. We then pick another open branch as the current branch and repeat Step 1. If no open branch exists, we stop and output `unsat`.
2. If a *propagation* rule (those with one conclusion) in $\mathcal{R}_1$ is applicable, apply one and go to Step 1.
3. If a *split* rule (those with more than one conclusion) in $\mathcal{R}_1$ is applicable, apply one and go to Step 1.
4. If GUESS EMPTY SET rule is applicable, apply it and go to Step 1.

file	output	time (s.)	# V	# L
vc1	unsat	0.00	3	3
vc2a	unsat	0.01	17	8
vc2b	sat	0.01	15	7
vc2	unsat	0.00	8	5
vc3a	unsat	0.00	6	0
vc3b	sat	0.01	17	8
vc3	unsat	0.00	6	0
vc4b	sat	0.22	45	16
vc4	unsat	0.07	57	18
vc5b	sat	1.71	71	22
vc5	unsat	0.36	68	21
vc6a	unsat	0.02	34	14
vc6b	sat	0.14	31	13
vc6c	sat	0.06	34	14
vc6	sat	0.02	38	18

(a) Jahob

file	output	time (s.)	# V	# L
vc1	1 sat/4 unsat	0.02	12	6
vc2	1 sat/3 unsat	0.07	39	23
vc3	2 sat/2 unsat	0.09	54	21
vc4	1 sat/3 unsat	0.02	0	0
vc5	2 sat/2 unsat	0.08	27	13
vc6	1 sat/3 unsat	0.01	0	0
vc7	2 sat/4 unsat	0.34	56	33
vc8	1 sat/3 unsat	0.01	0	0
vc9	2 sat/2 unsat	0.09	39	19
vc10	2 sat/2 unsat	0.32	94	32

(b) Leon

Fig. 10: Results on program verification benchmarks.

5. If an introduce or merge rule in $\mathcal{R}_2$ is applicable, apply it and go to Step 1.
6. If any of the remaining rules is applicable, apply one and go to Step 1.
7. At this point, the current branch is saturated. Stop and output `sat`.

Note that if there are no constraints involving the cardinality operator, then steps 1 to 3 above are sufficient for completeness.

5.2 Experimental evaluation

We evaluated our procedure on benchmarks obtained from verification of programs. The experiments were run on a machine with 3.40GHz Intel i7 CPU with a memory limit of 3 GB and timeout of 300 seconds. We used a development version of CVC4 for this evaluation.[11]Benchmarks are available on our website[12].

The first set of benchmarks consists of single query benchmarks obtained from verifying programs manipulating pointer-based data structures. These were generated by the Jahob system, and have been used to evaluate earlier work on decision procedures for finite sets and cardinality [10,11,13]. The results from running CVC4 on these benchmarks are provided in Figure 10a. The output reported by CVC4 is in the second column. The third column shows the solving time. The fourth and fifth columns give the maximum number of vertices (# V) and leaves[13] (# L) in the graph at any point during the run of the algorithm.

[11] Git commit `c833e17` at `https://github.com/CVC4/CVC4/commit/c833e176` .

[12] `http://cs.nyu.edu/~kshitij/setscard/`

[13] The # L statistic is updated only when explicitly computed, so the numbers are approximate. For the same reason, # L is 0 on certain benchmarks even though # V is not. This is because CVC4 was able to report unsat before the need for computing the set of leaves arose.

Keeping the number of leaves low is important to avoid a blowup from the MERGE EQUALITY II rule.

Although we have not rerun the algorithms from [10,11,13], we report here the experimental results as stated in the respective papers. As the experiments were run on different machines the comparison is only indicative, but it does suggest that our algorithm has comparable performance.

In [11], the procedure from [10] is reported to solve 12 of the 15 benchmarks with a timeout of 100 seconds, while the novel procedure in [11] is reported to solve 11 of the 15 benchmarks with the same timeout. The best-performing previous algorithm ([13]) can solve all 15 benchmarks in under a second.[14] As another point of comparison, we tested the algorithm from [13] on a benchmark of the type mentioned in Section 1.1: a single constraint of the form $x \sqsubseteq A_1 \sqcup \ldots \sqcup A_{21}$. As expected, the algorithm failed (it ran out of memory after 85 seconds). In contrast, CVC4 solves this problem instantaneously.

Finally, another important difference compared to earlier work is that our implementation is completely integrated in an actively developed and maintained solver, CVC4.[15] To highlight the usefulness of an implementation in a full-featured SMT solver, we did a second evaluation on a set of incremental (i.e., multiple-query) benchmarks obtained from the Leon verification system [4]. These contain a mix of membership and cardinality constraints together with the theories of datatypes and bitvectors. The results of this evaluation are shown in Figure 10b. The output column reports the number of sat and unsat queries in each benchmark. CVC4 successfully solves all of the queries in these benchmarks in under one second. To the best of our knowledge, no other SMT solver can handle this combination of theories.

6 Conclusion

We presented a new decision procedure for deciding finite sets with cardinality constraints and proved its correctness. A novel feature of the procedure is that it can reason directly and efficiently about both membership constraints and cardinality constraints. We have implemented the procedure in the CVC4 SMT solver, and demonstrated the feasibility as well as some advantages of our approach. We hope this work will enable the use of sets and cardinality in many new applications. We also expect to use it to drive the development of a standard theory of sets under the SMT-LIB initiative.

Acknowledgements. We thank the reviewers for their valuable and constructive suggestions. We thank Viktor Kuncak and Etienne Kneuss for valuable scientific discussions and for providing the Leon benchmarks. We thank Philippe Suter for his help running the algorithm from [13].

[14] Note that [13] includes a second set of benchmarks, but we were unable to evaluate our algorithm on these, as they were only made available in a non-standard format and were missing crucial datatype declarations.

[15] One reason we were unable to do a more thorough comparison with previous work is that those implementations are no longer being maintained.

References

1. Kshitij Bansal. *Decision Procedures for Finite Sets with Cardinality and Local Theory Extensions*. PhD thesis, New York University, January 2016.
2. Clark Barrett, Christopher Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.
3. Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185, chapter 26, pages 825–885. IOS Press, February 2009.
4. Régis William Blanc, Etienne Kneuss, Viktor Kuncak, and Philippe Suter. An overview of the Leon verification system: Verification by translation to recursive functions. In *Scala Workshop*, 2013.
5. D. Cantone, E. G. Omodeo, and A. Policriti. *Set Theory for Computing. From Decision Procedures to Logic Programming with Sets*. Monographs in Computer Science. Springer, 2001.
6. Domenico Cantone and Calogero G Zarba. A new fast tableau-based decision procedure for an unquantified fragment of set theory. In *Int. Workshop on First-Order Theorem Proving (FTP98)*, 1998.
7. Leonardo De Moura and Nikolaj Bjørner. Generalized, efficient array decision procedures. In *Formal Methods in Computer-Aided Design (FMCAD 2009)*, pages 45–52. IEEE, 2009.
8. Dejan Jovanović and Clark Barrett. Polite theories revisited. In *Proceedings of the* 17^{th} *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '10)*, volume 6397 of *LNCS*, pages 402–416. Springer, October 2010.
9. Daniel Kröning, Philipp Rümmer, and Georg Weissenbacher. A proposal for a theory of finite sets, lists, and maps for the SMT-LIB standard. In *Proceedings of the* 7^th *International Workshop on Satisfiability Modulo Theories (SMT '09)*, August 2009.
10. Viktor Kuncak, HuuHai Nguyen, and Martin Rinard. Deciding Boolean algebra with Presburger arithmetic. *Journal of Automated Reasoning*, 36(3):213–239, 2006.
11. Viktor Kuncak and Martin Rinard. Towards efficient satisfiability checking for Boolean Algebra with Presburger Arithmetic. In *Conference on Automated Deduction (CADE-21)*, volume 4603 of *Lecture Notes in Computer Science*. Springer, 2007.
12. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.
13. Philippe Suter, Robin Steiger, and Viktor Kuncak. Sets with cardinality constraints in Satisfiability Modulo Theories. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2011.
14. Calogero G. Zarba. Combining sets with integers. In *Frontiers of Combining Systems, 4th International Workshop, FroCoS 2002*, pages 103–116, 2002.