REASONING WITH FINITE SETS AND CARDINALITY CONSTRAINTS IN SMT

KSHITIJ BANSAL, CLARK BARRETT, ANDREW REYNOLDS, AND CESARE TINELLI

Google, Inc., 76 9th Ave, New York, NY, 10011, USA.
e-mail address: kbk@google.com

Department of Computer Science, Gates 341, Stanford University, Stanford, CA, 94305, USA.
e-mail address: barrett@cs.stanford.edu

Department of Computer Science, 14 MacLean Hall, The University of Iowa, Iowa City, IA, 52242, USA.
e-mail address: andrew-reynolds@uiowa.edu

Department of Computer Science, 14 MacLean Hall, The University of Iowa, Iowa City, IA, 52242, USA.
e-mail address: cesare-tinelli@uiowa.edu

Abstract. We consider the problem of deciding the satisfiability of quantifier-free formulas in the theory of finite sets with cardinality constraints. 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 in previous work. Instead, we develop a novel technique wherein potentially overlapping regions are considered incrementally as needed, using a graph to track the interaction among the different regions. The calculus has been designed to facilitate its implementation within SMT solvers based on the DPLL(T) architecture. Our experimental results demonstrate that the new techniques are competitive with previous techniques and can scale much better on certain classes of problems.

2012 ACM Subject Classification: Theory of computation: Automated reasoning.

Key words and phrases: Satisfiability modulo theories, Finite sets, Decision procedures.

This article extends [BRBT16] with additional calculus rules and a full proof of correctness.

This work was partially supported by NSF grants 1228765, 1228768, and 1320583. The first author was at New York University when this work was completed.

DOI:10.2168/LMCS-???
© Bansal et al.
Creative Commons

1. Introduction

Satisfiability modulo theories (SMT) solvers are at the heart of many formal methods tools. One of the reasons for their popularity is that fast, dedicated decision procedures for fragments of first-order logic that SMT solvers implement 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 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 finite set theory. Our main motivation is that sets are a common abstraction used in programming. As with other general-purpose SMT theories such as the theories of arrays and bitvectors, the theory of finite sets is useful for modeling a variety of program constructs. Sets are also used directly in high-level programming languages such as SETL [SDSD86] and in specification languages such as Alloy [Jac12], B [AA05] and Z [ASM80]. 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 for the theory of finite sets which can handle basic set operations, such as membership, union, intersection, and difference, and which can also reason efficiently about set cardinalities and linear constraints involving them. The calculus is explicitly designed for easy integration into the DPLL(T) framework [NOT06]. We briefly describe our implementation in the DPLL(T)-based SMT solver CVC4 and an initial experimental evaluation of this implementation.

1.1. Related work.

In the SMT community, the desire to support a theory of finite sets with cardinality goes at least as far back as a proposal by Kröning et al. [KRW09]. That article focuses on formalizing the semantics and representation of the theory within the context of the SMT-LIB standard, rather than on a decision procedure for deciding it.

There is a stream of research on exploring decidable fragments of set theory (often referred to in the literature as syllogistics) [COP01]. 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 introduced by Cantone and Zarba [CZ98]. The part of our calculus covering this fragment builds on their work. De Moura and Bjørner presented an extension of the theory of arrays [DMB09] that can be used to encode the MLSS fragment. However, this approach cannot be used to encode cardinality constraints.

In this paper, we consider an extension of the MLSS fragment with set cardinality operations, whose decidability was established by Zarba [Zar02, Zar05]. The decision procedure described by Zarba involves making an upfront guess that is exponential in the number of set variables, making it non-incremental and highly impractical. That said, the focus of that work is on establishing decidability and not on providing an efficient procedure.

Another closely related logical fragment is the Boolean Algebra and Presburger Arithmetic (BAPA) fragment, for which several algorithms have been proposed [KNR06, KR07, SSK11]. Though BAPA does not have the membership predicate or the singleton operator in its language, Suter et al. [SSK11, Section 4] show how one can generalize their algorithm for such reasoning. Intuitively, singleton sets can be simulated by imposing a cardinality constraint $\mathsf{card}(X) = 1$. Similarly, membership constraints of the form $x \sqsubseteq S$ can be encoded as $X \sqsubseteq S$ by introducing a singleton subset X. This reduction can lead to significant inefficiencies, however. Consider the following simple example: $x \sqsubseteq S_1 \sqcup (S_2 \sqcup (\ldots \sqcup (S_{99} \sqcup S_{100})))$. In our calculus, a straightforward repeated application of one of the rules for set unions can determine the satisfiability of this constraint. In contrast, in a reduction to BAPA, membership reasoning is reduced to reasoning about cardinalities of different sets. For example, the algorithm in [SSK11] will reduce the problem to an arithmetic problem 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 the 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 introduced lazily.

A procedure for cardinality constraints over multisets is considered in [PK08]. A recent procedure for reasoning about sets and measure functions is given by Bender et al [BS17], which also relies on a reduction from set reasoning to arithmetic reasoning. Reasoning about sets with cardinality constraints in the context of invariant checking for bounded model checking is considered by Alberti et al. [AGP16], and in the context of invariant synthesis by von Gleissenthall et el. [vGBR16]. These works too rely on reductions to arithmetic and do not involve the use of dedicated decision procedures for sets in SMT solvers. Other procedures for reasoning about sets include a unification-based approach by Cristiá et al. [CR16].

The theory we consider in this paper can be seen as the combination of Presburger arithmetic with the theory of finite sets, with the cardinality operator acting as a *bridging function* between the two theories. Decision procedures for non-disjoin combinations of theories with bridging functions have been studied by Sofronie-Stokkermans [SS09] and Chocron et al. [CFR15]. Their main contribution is the identification of restrictions on the theories and the development of combination methods that allow one to construct a decision procedure for the combined theory as a modular combination of the decision procedures for the component theories. That work is mostly limited to cases of bridging functions from the theory of algebraic datatypes to other theories, where the bridging function is definable by recursion over constructor terms. It does not apply to our setting because neither our source theory nor the bridging function match those requirements. Our approach is similar though in that it tries to separate as much as possible the reasoning about sets proper from the reasoning about their cardinality, so as to leverage off-the-shelf linear integer arithmetic solvers in order to reason about cardinalities.

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., [BSST09] for more details). Let Σ be a many-sorted signature. We use $\approx$ as the (infix) logical symbol for equality for all sorts in Σ and always interpret it as the

Constant and function symbols:

$n : \mathsf{Card}$ for all $n \in \mathbb{N}$ $\quad$ $\mathsf{-} : \mathsf{Card} \to \mathsf{Card}$ $\quad$ $\mathsf{+} : \mathsf{Card} \times \mathsf{Card} \to \mathsf{Card}$

$\emptyset : \mathsf{Set}$ $\quad$ $\mathsf{card} : \mathsf{Set} \to \mathsf{Card}$ $\quad$ $\{\cdot\} : \mathsf{Element} \to \mathsf{Set}$ $\quad$ $\sqcup, \sqcap, \setminus : \mathsf{Set} \times \mathsf{Set} \to \mathsf{Set}$

Predicate symbols:

$\mathsf{<} : \mathsf{Card} \times \mathsf{Card}$ $\quad$ $\mathsf{>=} : \mathsf{Card} \times \mathsf{Card}$ $\quad$ $\sqsubseteq : \mathsf{Set} \times \mathsf{Set}$ $\quad$ $\sqsubset : \mathsf{Element} \times \mathsf{Set}$

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

identity relation. 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 $\mathbf{I}$). We refer to $\mathbf{I}$ 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 $\mathcal{I}$ of T that satisfies one, there is a model of T that satisfies the other and differs from $\mathcal{I}$ 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 are interested in 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.[1] For simplicity here, we will describe instead a many-sorted theory of sets of sort Set whose elements are all of sort $\mathsf{Element}$. 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 $\mathsf{Element}$ sort with a sort in $\mathfrak{T}$ but with the restriction that the sort must be interpreted in $\mathfrak{T}$ as an infinite set.[2] Note that the many-sorted setting limits us to sets of elements of the same type (so sets such as $\{1, \{2,3\}, \{\{5\}\}\}$ are not representable). Also, we limit our language to consider only *flat* sets (i.e., no sets of sets of integers, say) although this restriction can be lifted by combining $\mathfrak{T}$ with (copies of) itself using Nelson-Oppen combination. More generally, an input having set constraints over multiple element types $T_1, \ldots, T_n$ can be handled by invoking n copies of our procedure for these sorts and combining them in the standard way. 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.

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

[2] An extension that allows the sort to be interpreted as finite by relying on polite combination [JB10] is left to future work.

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 $\mathsf{Element}$ 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,[3] and a cardinality operator ($\mathsf{card}(\cdot)$), all interpreted as expected. The signature includes also symbols for the cardinality comparison ($\mathsf{<}$, $\mathsf{>=}$), subset ($\sqsubseteq$) and membership ($\sqsubseteq\!\!\!\!-$) predicates.

We call *set term* any term of sort Set, and *cardinality term* any term of sort Card with no occurrences of $\mathsf{card}(\cdot)$. A *set constraint* is an atomic formula of the form $s \approx t$, $s \sqsubseteq t$, $e \sqsubseteq\!\!\!\!- s$ or their negation, with s and t set terms or of the form $\mathsf{card}(s)$, and e a term of sort $\mathsf{Element}$. A *cardinality constraint* is a [dis]equality $[\neg]c \approx d$ or an inequality $c \mathsf{<} d$ or $c \mathsf{>=} d$ 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 $\mathsf{Element}$. A $\mathfrak{T}_S$*-constraint* is a set, cardinality or element constraint. We write $u \not\approx v$ and $e \not\sqsubseteq\!\!\!\!- t$ respectively as an abbreviation of $\neg u \approx v$ and $\neg e \sqsubseteq\!\!\!\!- t$.

We use x, y for variables of sort $\mathsf{Element}$; 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, $\mathrm{Vars}(\mathcal{C})$ (respectively, $\mathrm{Terms}(\mathcal{C})$) 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 set term s a unique cardinality variable c_s.

We are interested in checking the satisfiability in $\mathfrak{T}_S$ of conjunctions of $\mathfrak{T}_S$-constraints. While this problem is decidable, it has high worst-case time complexity [Zar02]. So our efforts are in the direction of producing a solver for $\mathfrak{T}_S$-constraints 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 reasoning 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.

Restriction 3.1. For simplicity, we consider as input to the calculus only finite sets $\mathcal{C}$ of constraints whose set constraints are in *flat form*. The latter 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 equisatisfiability-preserving transformations all of these assumptions can be made without loss of generality [COP01, Chapter 10]. These transformations include intermediate steps that replace constraints of the form $s \sqsubseteq t$ with $s \approx (s \sqcap t)$. They also include steps that replace each occurrence i of the same term t in

[3]We will use $\emptyset$, $\{$, and $\}$ also to denote sets at the meta level. The difference between their two uses should be clear from context.

union, intersection or set difference terms by a fresh variable T_i while adding the equality constraint $T_i \approx t$.

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 $E(\mathcal{G})$.

Initial states have the form $\langle \mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0, \mathcal{G}_0 \rangle$ where $\mathcal{G}_0$ is the empty graph and $(\mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0)$ is a partition of a given set of constraints $\mathcal{C}$ satisfying Restriction 3.1.

Since cardinality constraints can be processed by a standard arithmetic solver, and element constraints by a simple equality solver,[4] 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 $\mathcal{S}_0$, $\mathcal{M}_0$, and $\mathcal{A}_0$ satisfy Restriction 3.1 and $\mathcal{G}_0$ is an empty graph, if and only if $\mathcal{S}_0 \cup \mathcal{M}_0 \cup \mathcal{A}_0$ 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 $\mathcal{A}$.

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 [CZ98], though with some key differences. First, the rules operate over a set $\mathcal{T}$ of terms with sort Set 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. For any set $\mathcal{C}$ of constraints, let $\mathrm{Terms}_\sigma(\mathcal{C})$ refer to terms of sort σ in $\mathcal{C}$, with $\mathrm{Terms}(\mathcal{C})$ denoting all terms in $\mathcal{C}$. We define the binary relation $\approx^*_{\mathcal{C}} \subseteq \mathrm{Terms}(\mathcal{C}) \times \mathrm{Terms}(\mathcal{C})$ to be the reflexive, symmetric, and transitive closure of the relation on terms induced by the equality constraints in $\mathcal{C}$. Now, we define

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

Union Down I
$$\frac{x \not\sqsubseteq s \sqcup t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \triangleleft (x \not\sqsubseteq s) \triangleleft (x \not\sqsubseteq t)}$$

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

Union Up I
$$\frac{x \not\sqsubseteq s \in \mathcal{S}^* \qquad x \not\sqsubseteq t \in \mathcal{S}^* \qquad s \sqcup t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \triangleleft (x \not\sqsubseteq s \sqcup t)}$$

Union Up II
$$\frac{x \sqsubseteq u \in \mathcal{S}^* \qquad u \in \{s, t\} \qquad s \sqcup t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \triangleleft (x \sqsubseteq s \sqcup t)}$$

Inter Down I
$$\frac{x \sqsubseteq s \sqcap t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \triangleleft (x \sqsubseteq s) \triangleleft (x \sqsubseteq t)}$$

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

Inter Up I
$$\frac{x \sqsubseteq s \in \mathcal{S}^* \qquad x \sqsubseteq t \in \mathcal{S}^* \qquad s \sqcap t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \triangleleft (x \sqsubseteq s \sqcap t)}$$

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

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

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

Set difference Down 1
$$\frac{x \sqsubseteq s \setminus t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \triangleleft (x \sqsubseteq s) \triangleleft (x \not\sqsubseteq t)}$$

Set difference Down 2
$$\frac{x \not\sqsubseteq s \setminus t \in \mathcal{S}^* \qquad x \sqsubseteq s \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \triangleleft (x \sqsubseteq t)}$$

Set difference Down 3
$$\frac{x \not\sqsubseteq s \setminus t \in \mathcal{S}^* \qquad x \not\sqsubseteq t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \triangleleft (x \not\sqsubseteq s)}$$

Set difference Up 1
$$\frac{x \sqsubseteq s \in \mathcal{S}^* \qquad x \not\sqsubseteq t \in \mathcal{S}^* \qquad s \setminus t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \triangleleft (x \sqsubseteq s \setminus t)}$$

Set difference Up 2
$$\frac{x \not\sqsubseteq s \in \mathcal{S}^* \qquad s \setminus t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \triangleleft (x \not\sqsubseteq s \setminus t)}$$

Set difference Up 3
$$\frac{x \sqsubseteq t \in \mathcal{S}^* \qquad s \setminus t \in \mathcal{T}}{\mathcal{S} := \mathcal{S} \triangleleft (x \not\sqsubseteq s \setminus t)}$$

Set difference split
$$\frac{s \setminus t \in \mathcal{T} \qquad x \sqsubseteq s \in \mathcal{S}^* \qquad x \sqsubseteq t \notin \mathcal{S}^* \qquad x \not\sqsubseteq t \notin \mathcal{S}^*}{\mathcal{S} := \mathcal{S} \triangleleft (x \sqsubseteq t) \;\parallel\; \mathcal{S} := \mathcal{S} \triangleleft (x \not\sqsubseteq t)}$$

Figure 2. Union, intersection, and set difference rules.

the following closures on the components $\mathcal{M}$ and $\mathcal{S}$ of a state $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G} \rangle$:

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

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

$$\textsc{Set Disequality}\ \frac{s \not\approx t \in \mathcal{S} \qquad \{x \in \mathcal{T} \mid x \sqsubseteq s, x \not\sqsubseteq t \in \mathcal{S}^*\} = \emptyset \qquad \{x \in \mathcal{T} \mid x \not\sqsubseteq s, x \sqsubseteq t \in \mathcal{S}^*\} = \emptyset}{\mathcal{S} := \mathcal{S} \triangleleft (y \sqsubseteq s) \triangleleft (y \not\sqsubseteq t) \quad \| \quad \mathcal{S} := \mathcal{S} \triangleleft (y \not\sqsubseteq s) \triangleleft (y \sqsubseteq t)}$$

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

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

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 binary operator $\triangleleft$ that takes as input a set $\mathcal{C}$ of constraints and a single constraint l. Intuitively, $\mathcal{C} \triangleleft (l)$ adds l to $\mathcal{C}$ only if l is not in $\mathcal{C}$'s closure. More precisely,

$$\mathcal{C} \triangleleft (l) = \begin{cases} \mathcal{C} & \text{if } l \in \mathcal{C}^* \\ \mathcal{C} \cup \{l\} & \text{otherwise} \end{cases} \tag{3.1}$$

The set of *relevant* terms, denoted by $\mathcal{T}$, for a state $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G} \rangle$ consists of all terms from $\mathcal{S}$ and $\mathcal{G}$, namely: $\text{Terms}(\mathcal{S}) \cup V(\mathcal{G})$.

Figure 2 shows the rules for reasoning about membership in unions, intersections, and differences. 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. The restriction $\{u, v\} = \{s, t\}$ in the premise of some of the rules cover all the various cases where s, say, is the same as t, different from t, the same as u, and the same as v. Figure 3 shows rules for singletons, disequalities, and contradictions. Note in particular that the SET DISEQUALITY rule introduces a fresh variable y, denoting an element that is in one set but not in the other.

Example 3.2. *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, described in Figure 6 and Figure 7, operate on the graph component of the current state. Their purpose is to modify the graph so as to capture the mutual dependencies between set and cardinality constraints. They are based on the observation that (*i*) the cardinality of two sets, and that of their union, intersection and set difference are inter-related; and (*ii*) if two set terms are asserted to be equal, their cardinalities must match.

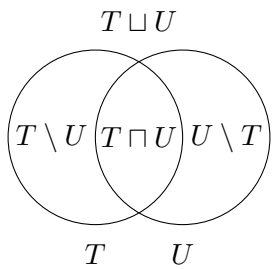

FIGURE 4. Venn regions for T and U.

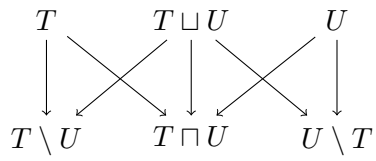

FIGURE 5. The same structure as a graph.

Figure 4 shows the Venn regions for two sets, T and U. The fact that $T \setminus U$, $T \sqcap U$ and $U \setminus T$ are disjoint imposes the following relationships between their cardinalities and those of T, U and $T \sqcup U$:

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

We can represent these same relationships using the graph in Figure 5. The nodes of the graph are set terms, and each node has the property of being the disjoint union of its children in the graph. Our calculus incrementally constructs a similar graph containing 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[5] 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 $s = T \setminus U$:
$V(\mathcal{G}') = V_2 = V(\mathcal{G}) \cup \{T, U, T \setminus U, T \sqcap U, U \setminus T\}$
$E(\mathcal{G}') = E_2 = E(\mathcal{G}) \cup \{(T, T \setminus U), (T, T \sqcap U), (U, T \sqcap U), (U, U \setminus T)\}$

(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 in the input set of constraints. It is not difficult to see that this property is preserved by every rule. This ensures that edges from a set variable node are added to the graph only once, maintaining the invariant that its children in the graph are disjoint. The only other rule which adds edges to the graph is the MERGE EQUALITY IIrule, but it only adds nodes from the leaves of the graph, creating a new set of disjoint leaves.

[5]See completeness proof in [Ban16, Chapter 2] for further details.

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

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

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

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

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

FIGURE 6. Graph extension rules.

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

$$\textsc{Merge Equality II}\quad \frac{s \approx t \in \mathcal{S} \qquad s, t \in V(\mathcal{G}) \qquad \mathcal{L}(s) \nsubseteq \mathcal{L}(t) \qquad \mathcal{L}(t) \nsubseteq \mathcal{L}(s)}{\mathcal{G} := \mathrm{merge}(\mathcal{G}, s, t)}$$

FIGURE 7. Merge rules.

Terms with explicit constraints on their cardinality are added to the graph by rule 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 explicitly 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 as follows. 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,

$$\mathcal{L}(n) = \left\{ n' \in \mathrm{Leaves}\,(n) \;\middle|\; n' \approx \emptyset \notin \mathcal{S}^* \right\}, \tag{3.2}$$

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

ARITHMETIC CONTRADICTION
$$\frac{\mathcal{A} \cup \hat{\mathcal{G}} \models_{\mathfrak{T}_A} \bot}{\mathsf{unsat}}$$

GUESS EMPTY SET
$$\frac{t \in \text{Leaves}\,(\mathcal{G})}{\mathcal{S} := \mathcal{S} \lhd (t \approx \emptyset) \quad \| \quad \mathcal{S} := \mathcal{S} \lhd (t \not\approx \emptyset)}$$

FIGURE 8. Additional graph rules.

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 $\mathcal{G}' = \text{merge}(\mathcal{G}, s, t)$ is defined as follows, where $L_1 = \mathcal{L}(s) \setminus \mathcal{L}(t)$ and $L_2 = \mathcal{L}(t) \setminus \mathcal{L}(s)$:

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

MERGE EQUALITY II introduces a quadratic number of leaves ($|L_1| \cdot |L_2|$). To reduce the impact of merge operations, a useful rule to apply early on is the (optional) GUESS EMPTY SET rule in Figure 8. It guesses if a leaf node is equal to the empty set or not. The use of this rule is illustrated in Example 3.3. Here and in Figure 9, $\text{Leaves}\,(\mathcal{G}) = \{v \in V(\mathcal{G}) \mid C(v) = \emptyset\}$.

In Figure 8 we denote by $\hat{\mathcal{G}}$ the collection of all of the following cardinality constraints imposed by graph $\mathcal{G}$:

(1) For each set term $s \in V(\mathcal{G})$, its cardinality (denoted by its corresponding cardinality variable c_s) is the sum of the cardinalities of its 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:
$$\{c_s \mathsf{>=} 0 \mid s \in V(\mathcal{G})\}$$
(3) Every 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 relies on the arithmetic solver to check whether the constraints in $\hat{\mathcal{G}}$ are inconsistent with the input cardinality constraints.

3.3. **Cardinality and membership interaction.** The rules in Figure 9 propagate consequences of set membership constraints to the state components $\mathcal{M}$ 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 $\mathcal{E}$. In the rules, for term t of sort Set, $t_{\mathcal{S}}$ denotes the set $\{[x]_{\mathcal{E}} \mid x \sqsubseteq t \in \mathcal{S}^*\}$ of equivalence classes of elements known to be in t. The notation $\mathcal{A} \Rrightarrow c_t \geq n$ means that $c_t \mathsf{>=} k \in \mathcal{A}$ for some concrete constant $k \geq n$.

Rule MEMBERS ARRANGEMENT is used to decide which element variables constrained to be in the same set t should be identified and which should not. Once applied to completion, Rule PROPAGATE MINSIZE can then be used to determine a lower bound for the cardinality of that set. The (optional) 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

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

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

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

FIGURE 9. Cardinality and membership interaction rules.

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

Example 3.3. *Consider again the constraints from Example 3.2, but now augmented with cardinality constraints:*

$$\begin{aligned} \mathcal{S} &= \{S \approx A \sqcup B,\, S \approx C \sqcap D,\, x \sqsubseteq C,\, x \not\sqsubseteq D,\, y \not\sqsubseteq S,\, y \sqsubseteq D\} \\ \mathcal{A} &= \{c_S \approx \mathsf{card}(S),\, c_C \approx \mathsf{card}(C),\, c_D \approx \mathsf{card}(D),\, c_S >= 4,\, c_C + c_D < 10\} \end{aligned}$$

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*). Node $A \sqcup B$ 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 the latter nodes respectively 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 cardinality constraints now include $c_S \approx c_{l_1} + c_{l_2} + c_{l_3}$, $c_C \approx c_{l_1} + c_{l_2} + c_{l_3} + c_{l_4}$, and $c_D \approx c_{l_1} + c_{l_2} + c_{l_3} + c_{l_5}$. 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 rules for set difference. Finally, we can use* PROPAGATE MINSIZE *to deduce $c_{l_4} >= 1$ and $c_{l_5} >= 1$. It is now not hard to see that using pure arithmetic reasoning, we can deduce that $c_C + c_D >= 10$ 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.

To prove these properties it is convenient to partition 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, with a root of the form $\langle \mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0, (\emptyset, \emptyset)\rangle$ where $\mathcal{S}_0 \cup \mathcal{M}_0 \cup \mathcal{A}_0$ satisfies Restriction 3.1 and the children of each non-root node are obtained by applying one of the derivation rules of the calculus to that node. Let $\mathcal{R}$ be a subset of the derivation rules of the calculus. A state is *saturated* with respect to $\mathcal{R}$ if no rules in $\mathcal{R}$ apply to it. A branch of a derivation tree is *closed* if it ends with unsat; it is *saturated* with respect to $\mathcal{R}$ if so is 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.

Definition 4.1 (Derivations)**.** Let $\mathcal{C}$ be a set of $\mathfrak{T}_S$-constraints. A *derivation (of $\mathcal{C}$)* 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 $\mathcal{S}_0 \cup \mathcal{M}_0 \cup \mathcal{A}_0 = \mathcal{C}$. A *refutation (of $\mathcal{C}$)* is a (finite) derivation of $\mathcal{C}$ that ends with a closed tree.

Remark. In the proofs below we implicitly rely on the fact that, for every state $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G}\rangle$ in a derivation tree, the constraints in $\mathcal{S} \cup \mathcal{M} \cup \mathcal{A}$ satisfy Restriction 3.1. This is the case because the restriction is imposed on root states and is preserved by all of its rules, as one can easily verify.

4.1. **Termination.**

Proposition 4.2 (Termination)**.** *Let $\mathcal{R}$ collect all rules in our calculus except for (the optional) rule* Guess Lower Bound*. Every derivation using only rules from $\mathcal{R}$ is finite.*

Proof. Let $\langle \mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0, (\emptyset, \emptyset)\rangle$ be the initial state of the derivation. We first define a well-founded relation $\succ$ over states. Next, we show that application of any rule in $\mathcal{R}$ to a leaf of a derivation tree gives smaller states with respect to this relation. As the relation is well-founded, it will follow that the derivation cannot be infinite.

In order to define $\succ$, we define f_i for $i \in \{1, 2, \ldots, 9\}$, each of which maps a state $\sigma = \langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G}\rangle$ to a natural number (non-negative integer). We denote the set of natural numbers by $\mathbb{N}$.

- $f_1(\sigma)$: 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 $\mathcal{L}(t_1) \neq \mathcal{L}(t_2)$.
- $f_2(\sigma)$: size of $(\mathrm{Terms}_{\mathsf{Set}}(\mathcal{S}) \cup \{\emptyset\}) \setminus V(\mathcal{G})$.
- $f_3(\sigma)$: size 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.
- $f_5(\sigma)$: size of $\mathrm{Terms}_{\mathsf{Set}}(\mathcal{S}) \cup \{\emptyset\} \cup V(\mathcal{G})$.
- $f_6(\sigma)$: size of $\mathrm{Terms}_{\mathsf{Element}}(\mathcal{S} \cup \mathcal{M})$.
- $f_7(\sigma)$: size of $\mathcal{M}^*$ subtracted from $2 \cdot (f_6(\sigma))^2$. As all constraints in $\mathcal{M}^*$ are either $x \approx y$ or $x \not\approx y$ with x and y in $\mathrm{Terms}_{\mathsf{Element}}(\mathcal{S} \cup \mathcal{M})$, the size of $\mathcal{M}^*$ can be at most $2 \cdot (f_6(\sigma))^2$. Thus, $f_7(\cdot)$ is well-defined as a map into $\mathbb{N}$.
- $f_8(\sigma)$: size of $\mathcal{S}^*$ subtracted from $2 \cdot (f_5(\sigma))^2 + 2 \cdot f_5(\sigma) \cdot f_6(\sigma)$. There are at most $2 \cdot (f_5(\sigma))^2$ constraints of the form $s \approx t$ or $s \not\approx t$ in $\mathcal{S}^*$ as s and t are in $\mathrm{Terms}_{\mathsf{Set}}(\mathcal{S}) \cup \{\emptyset\} \cup V(\mathcal{G})$. There are at most $2 \cdot f_5(\sigma) \cdot f_6(\sigma)$ constraints of the form $x \sqsubset s$ or $x \not\sqsubset s$ in $\mathcal{S}^*$ as x and s are in $\mathrm{Terms}_{\mathsf{Element}}(\mathcal{S} \cup \mathcal{M})$ and $\mathrm{Terms}_{\mathsf{Set}}(\mathcal{S}) \cup \{\emptyset\} \cup V(\mathcal{G})$ respectively. Thus, $f_8(\cdot)$ is well-defined as a map into $\mathbb{N}$.

- $f_9(\sigma)$: size of $(\text{Terms}_{\mathsf{Set}}(\mathcal{S}) \cup \{\emptyset\} \cup V(\mathcal{G})) \setminus \{t \in \text{Leaves}\,(\mathcal{G}) \mid \mathcal{A} \not\Rightarrow c_t \geq \mathsf{card}(t_{\mathcal{S}})\}$.

Then, we define the order $\succ$ over states as follows:

- $\sigma \succ \sigma'$ if $\sigma \neq \mathsf{unsat}$ and $\sigma' = \mathsf{unsat}$.
- $\sigma \succ \sigma'$ if $\sigma \neq \mathsf{unsat}$, $\sigma' \neq \mathsf{unsat}$, and
$$(f_1(\sigma), \ldots, f_9(\sigma)) >^9_{\mathsf{lex}} \big(f_1(\sigma'), \ldots, f_9(\sigma')\big)$$
where $\big(\mathbb{N}^9, >^9_{\mathsf{lex}}\big)$ is the 9-fold lexicographic product of ordering over natural numbers $(\mathbb{N}, >)$.
- $\sigma \not\succ \sigma'$ otherwise.

The well-foundedness of $\succ$ over states follows from the well-foundedness of $\big(\mathbb{N}^9, >^9_{\mathsf{lex}}\big)$ [BN98, Section 2.4].

Let $r \in \mathcal{R}$ be a rule applicable at state σ, and let σ' be the state after the application of the rule (if there are multiple conclusions, denote the state on first branch as σ'_1, second branch as σ'_2 and so on). We note below for each rule $r \in \mathcal{R}$ the relation between $f_1(\sigma)$, …, $f_9(\sigma)$ and $f_1(\sigma')$, …, $f_9(\sigma')$ which establishes that $\sigma \succ \sigma'$.

- First, we consider Rules for intersection (Figure 2), union (Figure 2), set difference (Figure 2) and Rule Singleton for singleton. None of these rules introduce equalities of set terms, nor do they affect the graph $\mathcal{G}$; thus $f_1(\sigma) \geq f_1(\sigma')$. The only terms introduced to $\mathcal{S}$ are from $V(\mathcal{G})$, thus $f_2(\sigma) = f_2(\sigma')$. None of these rules update $\mathcal{G}$ or introduce equalities or disequalities of set terms, thus $f_3(\sigma) = f_3(\sigma')$. None of these rules introduce disequalities between set terms, thus $f_4(\sigma) \geq f_4(\sigma')$. None of these rules introduce set terms not already in $\mathcal{S}$ or $V(\mathcal{G})$, thus $f_5(\sigma) = f_5(\sigma')$. None of the rules introduce $\mathsf{Element}$ variables not already in $\mathcal{S}$ or $\mathcal{M}$, thus $f_6(\sigma) = f_6(\sigma')$. None of these rules update $\mathcal{M}$, thus $f_7(\sigma) = f_7(\sigma')$.

  Each of these rules updates $\mathcal{S}$. Recall that for a rule to be applicable at σ, the resulting state must be different from σ. From the definition of $\lhd$, we can conclude that size of $\mathcal{S}^*$ has increased. As $f_5(\sigma) = f_5(\sigma')$ and $f_6(\sigma) = f_6(\sigma')$, it follows that $f_8(\sigma) > f_8(\sigma')$.
- Next, we consider Rules Single Member, Single Non-member and Members Arrangement. None of these rules introduce equalities of set terms, thus $f_1(\sigma) \geq f_1(\sigma')$. None of these rules introduce set terms to $\mathcal{S}$ or $V(\mathcal{G})$, thus $f_2(\sigma) = f_2(\sigma')$. None of these rules update $\mathcal{G}$ or introduce equalities or disequality of set terms, thus $f_3(\sigma) = f_3(\sigma')$. None of these rules introduce disequalities of set terms, thus $f_4(\sigma) \geq f_4(\sigma')$. None of these rules introduce set terms to $\mathcal{S}$ or $V(\mathcal{G})$, thus $f_5(\sigma) = f_5(\sigma')$. None of the rules introduce $\mathsf{Element}$ variables not already in $\mathcal{S}$ or $\mathcal{M}$, thus $f_6(\sigma) = f_6(\sigma')$.

  Each of these rules updates $\mathcal{M}$. From the definition of $\lhd$, we can conclude that size of $\mathcal{M}^*$ has increased. As $f_6(\sigma) = f_6(\sigma')$, we can conclude that $f_7(\sigma) > f_7(\sigma')$.
- Next, we consider Rule Set Disequality. The rule does not introduce any equality of set terms, thus $f_1(\sigma) \geq f_1(\sigma'_i)$ for $i \in \{1, 2\}$. The rule does not introduce set terms to $\mathcal{S}$ or $V(\mathcal{G})$, thus $f_2(\sigma) = f_2(\sigma'_i)$ for $i \in \{1, 2\}$. The rule does not update $\mathcal{G}$, thus $f_3(\sigma) \geq f_3(\sigma'_i)$ for $i \in \{1, 2\}$. The premise of the rule does not hold after application of the rule on either of the branches. It follows that $f_4(\sigma) > f_4(\sigma'_i)$ for $i \in \{1, 2\}$.

- Next, we consider Introduce Rules (Figure 6). Note that none of these rules introduce equalities of set terms. Also note that if $t_1 \in V(\mathcal{G})$, $t_2 \in V(\mathcal{G})$ and $t_1 \approx t_2$ in $\mathcal{S}$ then $\mathcal{L}(t_1) = \mathcal{L}(t_2)$ (see Proposition 4.4, property 1). Thus, $f_1(\sigma) \geq f_1(\sigma')$.

  Each of the rules adds at least one new node to $\mathcal{G}$ which is in $\text{Terms}_{\mathsf{Set}}(\mathcal{S}) \cup \{\emptyset\}$. At the same time, $\mathcal{S}$ is unchanged. It follows that $f_2(\sigma) > f_2(\sigma')$.
- Rules Merge Equality I and Merge Equality II. Though these rules add equalities of the form $u \approx \emptyset$ to $\mathcal{S}$, the equalities are such that $u \in V(\mathcal{G})$, $\emptyset \in V(\mathcal{G})$ and $\mathcal{L}(u) = \emptyset = \mathcal{L}(\emptyset)$. It follows that $f_1(\sigma) \geq f_1(\sigma')$.

  Now, observe that for Rule Merge Equality I or Rule Merge Equality II to be applicable, there must exist $s \approx t \in \mathcal{S}$ such that $\mathcal{L}(s) \neq \mathcal{L}(t)$. After the application of the rule, $\mathcal{L}(s) = \mathcal{L}(t)$. This shows that $f_1(\sigma) > f_1(\sigma')$.
- Rule Merge Equality III. For the rule to be applicable, there must exist $s \approx t \in \mathcal{S}$ such that $\mathcal{L}(s) \neq \mathcal{L}(t)$. After the application of the rule, $\mathcal{L}(s) = \mathcal{L}(t)$. Thus, necessarily $f_1(\sigma) > f_1(\sigma')$.
- Rule Guess Empty Set. Note that though this rule may add an equality of the form $t \approx \emptyset$ on the first branch, using the same reasoning as for Rules Merge Equality I and Merge Equality II above, we can conclude that $f_1(\sigma) \geq f_1(\sigma'_1)$. On the second branch, as no disequality is added, we get that $f_1(\sigma) \geq f_1(\sigma'_2)$. Only terms introduced to $\mathcal{S}$ are from $V(\mathcal{G})$, thus $f_2(\sigma) = f_2(\sigma'_1)$ for $i \in \{1, 2\}$.

  In order to apply the rule, we pick a $t \in \text{Leaves}\,(G)$ such that $t \approx \emptyset \notin \mathcal{S}^*$ and $t \not\approx \emptyset \notin \mathcal{S}^*$. On the first branch, $t \approx \emptyset \in \mathcal{S}^*$, thus $f_3(\sigma) > f_3(\sigma'_1)$. On the second branch, $t \not\approx \emptyset \in \mathcal{S}^*$, thus $f_3(\sigma) > f_3(\sigma'_2)$.
- Rule Propagate Minsize. The rule does not update $\mathcal{S}$, $\mathcal{M}$, or $\mathcal{G}$, thus $f_1(\sigma) = f_1(\sigma')$, $f_2(\sigma) = f_2(\sigma')$, $f_3(\sigma) = f_3(\sigma')$, $f_4(\sigma) = f_4(\sigma')$, $f_5(\sigma) = f_5(\sigma')$, $f_6(\sigma) = f_6(\sigma')$, $f_7(\sigma) = f_7(\sigma')$, and $f_8(\sigma) = f_8(\sigma')$. But, $f_9(\sigma) > f_9(\sigma')$.
- Rules Eq Unsat, Set Unsat, Empty Unsat, and Arithmetic contradiction. For each of these rules to be applicable, $\sigma \neq \mathsf{unsat}$. On the other hand, $\sigma' = \mathsf{unsat}$ after the application of the rule. By definition, $\sigma \succ \sigma'$.

□

Remark. It is easy to extend the termination proof above to include the optional rule Guess Lower Bound. It would involve tracking sizes of additional objects—a strategy similar to one adopted for Rule Guess Empty Set in our proof would suffice.

4.2. **Completeness.** We prove properties about different subsets of rules, developing the completeness proof in stages. We start with a proposition about rule set $\mathcal{R}_1$.

Proposition 4.3. *Let $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G} \rangle$ be a derivation tree leaf that is saturated with respect to $\mathcal{R}_1$. 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 $x, y \in \text{Vars}\,(\mathcal{M}) \cup \text{Vars}\,(\mathcal{S})$ of sort Element*,*
$x^{\mathfrak{S}} = y^{\mathfrak{S}}$ *if and only if* $x \approx y \in \mathcal{M}^*$.
(2) For all $S \in \text{Vars}\,(\mathcal{S})$ of sort Set*, $S^{\mathfrak{S}} = \left\{ x^{\mathfrak{S}} \;\middle|\; x \sqsubseteq S \in \mathcal{S}^* \right\}$.*
(3) For all $c_S \in \text{Vars}\,(\mathcal{S})$ of sort Card*, $c_S^{\mathfrak{S}} = \left| S^{\mathfrak{S}} \right|$.*

Proof. Since the models of $\mathfrak{T}_S$ are closed under variable reassignment, we pick an arbitrary model $\mathfrak{S}$ of $\mathfrak{T}_S$ and show that we can change its interpretation of the variables of $\mathcal{S} \cup \mathcal{M}$ to satisfy the properties above.

We start by interpreting all variables of Element sort in $\mathcal{S} \cup \mathcal{M}$ so that, for all x and y in $\mathrm{Vars}(\mathcal{M}) \cup \mathrm{Vars}(\mathcal{S})$ of Element sort,

$$x^{\mathfrak{S}} = y^{\mathfrak{S}} \text{ if and only } x \approx y \in \mathcal{M}^* \ .$$

It follows that $\mathfrak{S}$ satisfies $\mathcal{M}$. Next, let $\mathfrak{S}$ interpret each variable S of Set sort in $\mathrm{Vars}(\mathcal{S})$ as:

$$S^{\mathfrak{S}} = \left\{ x^{\mathfrak{S}} \;\middle|\; x \sqsubseteq S \in \mathcal{S}^* \right\}$$

and each variable c_S of Card sort in $\mathrm{Vars}(\mathcal{S})$ as:

$$c_S^{\mathfrak{S}} = \left| S^{\mathfrak{S}} \right| \ .$$

For any set term s, define

$$\mathrm{Elements}(s) = \left\{ x^{\mathfrak{S}} \;\middle|\; x \sqsubseteq s \in \mathcal{S}^* \right\} \ . \tag{4.1}$$

Let $\mathcal{T}$ be an arbitrary set of set terms which includes all set terms in $\mathcal{S}$. Using the assumption that the given state is saturated, we show by structural induction on set terms that for any set term $s \in \mathcal{T}$:

$$\mathrm{Elements}(s) = s^{\mathfrak{S}} \tag{4.2}$$

Case 1 (s is a variable). The definition of $\mathrm{Elements}(s)$ is identical to that of $s^{\mathfrak{S}}$.

Case 2 (s is $\emptyset$). Rule Empty Unsat would apply to the state if there was a constraint of the form $x \sqsubseteq \emptyset$ in $\mathcal{S}^*$. It follows that $\mathrm{Elements}(\emptyset) = \emptyset$.

Case 3 (s is $\{x\}$). As $s^{\mathfrak{S}} = \left\{x^{\mathfrak{S}}\right\}$, it is sufficient to show that $\mathrm{Elements}(s) = \left\{x^{\mathfrak{S}}\right\}$. Since rule Singleton is not applicable, we can conclude that $x \sqsubseteq s \in \mathcal{S}^*$. It follows that $\left\{x^{\mathfrak{S}}\right\} \subseteq \mathrm{Elements}(s)$. The other direction, $\mathrm{Elements}(s) \subseteq \left\{x^{\mathfrak{S}}\right\}$, follows because of saturation with respect to rule Single Member:

$e \in \mathrm{Elements}(\{x\})$	
$e = y^{\mathfrak{S}}$ for some y with $y \sqsubseteq \{x\} \in \mathcal{S}^*$	(definition)
$y \approx x \in \mathcal{M}^*$	Rule Single Member
$y^{\mathfrak{S}} = x^{\mathfrak{S}}$	($\mathfrak{S}$ satisfies $\mathcal{M}$)
$e \in \left\{x^{\mathfrak{S}}\right\}$	($e = y^{\mathfrak{S}}$)

Case 4 (s is $t \sqcap u$). We need to show $\mathrm{Elements}(t \sqcap u) = t^{\mathfrak{S}} \cap u^{\mathfrak{S}}$. The proof of the left-to-right inclusion depends on rule Inter Down I:

$e \in \mathrm{Elements}(t \sqcap u)$	
$e = x^{\mathfrak{S}}$ for some x with $x \sqsubseteq t \sqcap u \in \mathcal{S}^*$	(definition)
$x \sqsubseteq t \in \mathcal{S}^*$ and $x \sqsubseteq u \in \mathcal{S}^*$	(Rule Inter Down I)
$x^{\mathfrak{S}} \in \mathrm{Elements}(t)$ and $x^{\mathfrak{S}} \in \mathrm{Elements}(u)$	(definition)
$x^{\mathfrak{S}} \in t^{\mathfrak{S}}$ and $x^{\mathfrak{S}} \in u^{\mathfrak{S}}$	(induction)
$e \in t^{\mathfrak{S}} \cap u^{\mathfrak{S}}$	

For the other direction, $t^{\mathfrak{S}} \cap u^{\mathfrak{S}} \subseteq \text{Elements}(t \sqcap u)$, we rely on rule Inter Up I:

$e \in t^{\mathfrak{S}} \cap u^{\mathfrak{S}}$

$e \in t^{\mathfrak{S}}$ and $e \in u^{\mathfrak{S}}$

$e \in \text{Elements}(t)$ and $e \in \text{Elements}(u)$ (induction)

$x \sqsubseteq t \in \mathcal{S}^*$ and $y \sqsubseteq u \in \mathcal{S}^*$ with $x^{\mathfrak{I}} = y^{\mathfrak{I}} = e$ (definition)

$x \sqsubseteq t \in \mathcal{S}^*$ and $y \sqsubseteq u \in \mathcal{S}^*$ with $x \approx y \in \mathcal{M}^*$ (by construction)

$x \sqsubseteq t \in \mathcal{S}^*$ and $x \sqsubseteq u \in \mathcal{S}^*$

$x \sqsubseteq t \sqcap u \in \mathcal{S}^*$ ($t \sqcap u \in \mathcal{T}$, Rule Inter Up I)

$e \in \text{Elements}(t \sqcap u)$

Case 5 (s is $t \sqcup u$). First we show that $\text{Elements}(t \sqcup u) \subseteq t^{\mathfrak{S}} \cap u^{\mathfrak{S}}$:

$e \in \text{Elements}(t \sqcup u)$

$e = x^{\mathfrak{S}}$ for some x with $x \sqsubseteq t \sqcup u \in \mathcal{S}^*$ (definition)

$x \sqsubseteq t \in \mathcal{S}^*$ or $x \sqsubseteq u \in \mathcal{S}^*$ (Rule Union split)

$x^{\mathfrak{S}} \in \text{Elements}(t)$ or $x^{\mathfrak{S}} \in \text{Elements}(u)$ (definition)

$x^{\mathfrak{S}} \in t^{\mathfrak{S}}$ or $x^{\mathfrak{S}} \in u^{\mathfrak{S}}$ (induction)

$e \in t^{\mathfrak{S}} \cup u^{\mathfrak{S}}$

Then we show that $t^{\mathfrak{S}} \cup u^{\mathfrak{S}} \subseteq \text{Elements}(t \sqcup u)$:

$e \in t^{\mathfrak{S}} \cup u^{\mathfrak{S}}$

$e \in t^{\mathfrak{S}}$ or $e \in u^{\mathfrak{S}}$

$e \in \text{Elements}(t)$ or $e \in \text{Elements}(u)$ (induction)

$x \sqsubseteq t \in \mathcal{S}^*$ or $x \sqsubseteq u \in \mathcal{S}^*$ where $x^{\mathfrak{I}} = e$ (definition)

$x \sqsubseteq t \sqcup u \in \mathcal{S}^*$ ($t \sqcup u \in \mathcal{T}$, Rule Union Up II)

$e \in \text{Elements}(t \sqcap u)$

Case 6 (s is $t \setminus u$). First we show that $\text{Elements}(t \setminus u) \subseteq t^{\mathfrak{S}} \setminus u^{\mathfrak{S}}$:

$e \in \text{Elements}(t \setminus u)$

$e = x^{\mathfrak{S}}$ for some x with $x \sqsubseteq t \setminus u \in \mathcal{S}^*$ (definition)

$x \sqsubseteq t \in \mathcal{S}^*$ and $x \not\sqsubseteq u \in \mathcal{S}^*$ (Rule Set difference Down 1)

From $x \not\sqsubseteq u \in \mathcal{S}^*$ we can conclude that $x^{\mathfrak{S}} \in \text{Elements}(u)$. In fact, if we assume otherwise, we have that $y \sqsubseteq u \in \mathcal{S}^*$ for some y with $x^{\mathfrak{S}} = y^{\mathfrak{S}}$. But then $x \approx y \in \mathcal{M}^*$ which implies that $x \sqsubseteq u \in \mathcal{S}^*$. This, however, make rules Set Unsat applicable. Then we have:

$x^{\mathfrak{S}} \in \text{Elements}(t)$ and $x^{\mathfrak{S}} \notin \text{Elements}(u)$

$x^{\mathfrak{S}} \in t^{\mathfrak{S}}$ and $x^{\mathfrak{S}} \notin u^{\mathfrak{S}}$ (induction)

$e \in t^{\mathfrak{S}} \setminus u^{\mathfrak{S}}$

We now show that $t^{\mathfrak{S}} \setminus u^{\mathfrak{S}} \subseteq \text{Elements}(t \setminus u)$:

$$
\begin{aligned}
& e \in t^{\mathfrak{S}} \setminus u^{\mathfrak{S}} \\
& e \in t^{\mathfrak{S}} \text{ and } e \notin u^{\mathfrak{S}} \\
& e \in \text{Elements}(t) \text{ and } e \notin \text{Elements}(u) && \text{(induction)} \\
& x \sqsubseteq t \in \mathcal{S}^* \text{ and } x \sqsubseteq u \notin \mathcal{S}^* \text{ for some } x \text{ with } x^{\mathfrak{I}} = e && \text{(definition)}
\end{aligned}
$$

We show by contradiction that $x \not\sqsubseteq u \in \mathcal{S}^*$. Assume the otherwise. Since $x \sqsubseteq u \notin \mathcal{S}^*$ and $t \setminus u \in \mathcal{T}$, the premise of rule Set difference split is satisfied. As we had neither $x \sqsubseteq u \in \mathcal{S}^*$ nor $x \not\sqsubseteq u \in \mathcal{S}^*$, we get a contradiction.

$$
\begin{aligned}
& x \sqsubseteq t \in \mathcal{S}^* \text{ and } x \not\sqsubseteq u \in \mathcal{S}^* && \text{(Rule Set difference split)} \\
& x \sqsubseteq t \setminus u \in \mathcal{S}^* && (t \setminus u \in \mathcal{T}, \text{ Rule Set difference Up 1}) \\
& e \in \text{Elements}(t \setminus u)
\end{aligned}
$$

Having established the property of Elements$(\cdot)$, showing that each constraint in $\mathcal{S}$ is satisfied by $\mathfrak{S}$ is straightforward:

(1) Let $x \sqsubseteq s \in \mathcal{S}$. Then, $x^{\mathfrak{S}} \in \text{Elements}(s)$ by (4.1) and $x^{\mathfrak{S}} \in s^{\mathfrak{S}}$ by (4.2).
(2) Let $x \not\sqsubseteq s \in \mathcal{S}$. We show $x^{\mathfrak{S}} \notin s^{\mathfrak{S}}$ by contradiction.

$$
\begin{aligned}
& x^{\mathfrak{S}} \in s^{\mathfrak{S}} && \text{(assume)} \\
& x^{\mathfrak{S}} \in \text{Elements}(s) && \text{(proved above)} \\
& x^{\mathfrak{S}} = y^{\mathfrak{S}} \text{ for some } y \text{ with } y \sqsubseteq s \in \mathcal{S}^* && \text{(definition)} \\
& x \approx y \in \mathcal{M}^* && (x^{\mathfrak{S}} = y^{\mathfrak{S}} \text{ iff } x \approx y \in \mathcal{M}^*) \\
& x \sqsubseteq s \in \mathcal{S}^* && \text{(definition of } \mathcal{S}^*) \\
& \text{Tableau is closed, contradiction.} && \text{(Rule Set Unsat)}
\end{aligned}
$$

(3) Let $s \approx t \in \mathcal{S}$. From the definition of $\mathcal{S}^*$ it follows that $\text{Elements}(s) = \text{Elements}(t)$. Since $s^{\mathfrak{S}} = \text{Elements}(s)$ and $t^{\mathfrak{S}} = \text{Elements}(t)$, it follows that $s^{\mathfrak{S}} = t^{\mathfrak{S}}$.
(4) Let $s \not\approx t \in \mathcal{S}$. From rule Set Disequality, it follows that there exists x such that either $x \sqsubseteq s \in \mathcal{S}^*$ and $x \not\sqsubseteq t \in \mathcal{S}^*$, or $x \not\sqsubseteq s \in \mathcal{S}^*$ and $x \sqsubseteq t \in \mathcal{S}^*$. It follows that either $x^{\mathfrak{S}} \in s^{\mathfrak{S}}$ and $x^{\mathfrak{S}} \notin t^{\mathfrak{S}}$, or $x^{\mathfrak{S}} \notin s^{\mathfrak{S}}$ and $x^{\mathfrak{S}} \in t^{\mathfrak{S}}$. In either case, we can conclude that $s^{\mathfrak{S}} \neq t^{\mathfrak{S}}$.
(5) Let $c_S \approx \mathsf{card}(S) \in \mathcal{S}$. By definition, both $c_S^{\mathfrak{S}} = \left|S^{\mathfrak{S}}\right| = \mathsf{card}(S)^{\mathfrak{S}}$.

□

For the next two results, let $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G} \rangle$ be a derivation tree leaf 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 4.4. *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) $\{t \approx u \mid t \approx u \in \mathcal{S}^*\} \models_{\mathfrak{T}_S} s \approx \bigsqcup_{t \in \mathcal{L}(s)} t$.[6]

Proof (Proposition 4.4, property 1). Let $s \approx t \in \mathcal{S}$, with $s \in V(\mathcal{G})$ or $t \in V(\mathcal{G})$. From rule Introduce Eq Right and rule Introduce Eq Left it follows that both $s \in V(\mathcal{G})$ and $t \in V(\mathcal{G})$. For each of the Rules Merge Equality I, Merge Equality II, and Merge Equality III; we show that after the application of the rule, $\mathcal{L}(s)$ and $\mathcal{L}(t)$ are equal.

Consider rule Merge Equality I. Let L_s and L_t denote $\mathcal{L}(s)$ and $\mathcal{L}(t)$ respectively before application of the rule. Let L'_s and L'_t denote $\mathcal{L}(s)$ and $\mathcal{L}(t)$ after application of the rule. For the rule to be applicable $L_s \subsetneq L_t$. The rule adds constraints to $\mathcal{S}$ so that $L'_t = L_t \setminus (L_t \setminus L_s)$. Equivalently, $L'_t = L_t \cap L_s = L_s$. Since $L'_s = L_s$, we get $L'_s = L_s = L'_t$.

The case for rule Merge Equality II is analogous to rule Merge Equality I.

Consider rule Merge Equality III. Let L_s and L_t denote $\mathcal{L}(s)$ and $\mathcal{L}(t)$ respectively before application of the rule. Let L'_s and L'_t denote $\mathcal{L}(s)$ and $\mathcal{L}(t)$ after application of the rule. Let $n \in L'_s$. Note that the merge operation only adds nodes and vertices. Thus, n is one of the following:

- $l_1 \sqcap l_2$ with $l_1 \in L_s$ and $l_2 \in L_t$: Since $(l_1, l_1 \sqcap l_2)$ as well as $(l_2, l_1 \sqcap l_2)$ is an edge, it follows that $n \in L'_t$.
- $l_1 \in L_s$. Since nodes in $L_s \setminus L_t$ have an outgoing edge, it must be the case that $l_1 \in L_s \cap L_t$. It follows that $n \in L'_t$.

This shows that $L'_s \subseteq L'_t$. The reasoning for $L'_t \subseteq L'_s$ is symmetrical.

As $s \approx t$, $s \in V(\mathcal{G})$, and $t \in V(\mathcal{G})$, the premise of at least once of the rules (Merge Equality I), (Merge Equality II), and (Merge Equality III) must be satisfied whenever $\mathcal{L}(s) \neq \mathcal{L}(t)$. As the branch is saturated, $\mathcal{L}(s) = \mathcal{L}(t)$ follows. □

Proof (Proposition 4.4, properties 2, 3, 4). As $\mathcal{D}$ is obtained from a derivation starting with a state with an empty graph, it is sufficient to show the properties hold for the empty graph, and that they are preserved each time the graph is modified by one of the rules.

The properties hold trivially for the empty graph. The interesting cases are when edges are added to the graph: i) add of a union, intersection, or set minus term, and ii) merge operation.

Observe that when we introduce $T \sqcup U$, $T \sqcap U$, and $T \setminus U$ to the graph, the following holds:

- $\text{Leaves}\,(T) = \{T \setminus U, T \sqcap U\}$,
- $\text{Leaves}\,(U) = \{T \sqcap U, U \setminus T\}$,
- $\text{Leaves}\,(T \sqcup U) = \{T \setminus U, T \sqcap U, U \setminus T\}$,
- $\text{Leaves}\,(T \sqcap U) = \{T \sqcap U\}$,
- $\text{Leaves}\,(T \setminus U) = \{T \setminus U\}$, and
- $\text{Leaves}\,(U \setminus T) = \{U \setminus T\}$.

We conclude that:

- $\text{Leaves}\,(T \sqcup U) = \text{Leaves}\,(T) \cup \text{Leaves}\,(U)$
- $\text{Leaves}\,(T \sqcap U) = \text{Leaves}\,(T) \cap \text{Leaves}\,(U)$
- $\text{Leaves}\,(T \setminus U) = \text{Leaves}\,(T) \setminus \text{Leaves}\,(U)$
- $\text{Leaves}\,(U \setminus T) = \text{Leaves}\,(U) \setminus \text{Leaves}\,(T)$

when an introduce rule is applied. Note that the merge operation only adds edges from existing leaf nodes, ensuring that the property is maintained by any application of merge.

[6] Technically, $\bigsqcup_{\ldots}$ is ambiguous. However, since $\sqcup$ is associative in $\mathfrak{T}_S$, bracketing does not matter in this context.

$\mathcal{L}(\cdot)$, as defined in (3.2), can also be defined as:

$$\mathcal{L}(n) = \text{Leaves}\,(n) \setminus E \tag{4.3}$$

where $E = \{n' \in V(\mathcal{G}) \mid n' \approx \emptyset \in \mathcal{S}^*\}$ does not depend on n. The properties in the proposition about $\mathcal{L}(\cdot)$ follow from the corresponding property of Leaves $(\cdot)$ just established, and above formulation of $\mathcal{L}(\cdot)$. □

Proof (Proposition 4.4, properties 5,6). The properties holds trivially for the empty graph.

Let $\mathcal{G}$ be the graph constraints. Let $s \in V(\mathcal{G})$. Let $s' \approx \emptyset$ be a new constraint such that $s' \in \mathcal{L}(s)$. Then, this modifies $\mathcal{L}(s)$, and we need to verify the Property 6 still holds. Note that for any structure in $\mathfrak{T}_S$, if s' is interpreted as empty set, the interpretation of $\bigsqcup_{t \in \mathcal{L}(s) \setminus \{s'\}} t$ will be same as $\bigsqcup_{t \in \mathcal{L}(s)} t$. Thus, if $s' \in \mathcal{L}(s)$ and

$$\models_{\mathfrak{T}_S} \left(\bigwedge_{P \in E} P \right) \Rightarrow \left(s \approx \bigsqcup_{t \in \mathcal{L}(s)} t \right) \quad ,$$

then

$$\models_{\mathfrak{T}_S} \left(s' \approx \emptyset \wedge \bigwedge_{P \in E} P \right) \Rightarrow \left(s \approx \bigsqcup_{t \in \mathcal{L}(s) \setminus \{s'\}} t \right) \quad .$$

It follows if $s' \approx \emptyset$ is added to $\mathcal{S}^*$ by a rule, the property 6 continue to hold. Also note that an equality is not removed by any rule (if there was such a rule, we would need to check the property continues to hold when the left side of the implication is weakened).

The only other rules which affect the properties are those which modify the graph directly, i.e. the add and merge operations.

We show that if $\mathcal{G}$ satisfies the properties, then so does add$(\mathcal{G}, s)$:

- s is $\emptyset$, S or $\{x\}$: trivially, as no edges are added.
- s is $T \sqcap U$: Note that because of the assumptions on the normal form, either $T \sqcap U$ already in the graph and add operation does not modify the graph, or it will add the nodes T, U, $T \setminus U$, $T \sqcap U$, and $U \setminus T$ to the graph, and edges between them. It is easy to see that the property 5 follows from:

  $$\models_{\mathfrak{T}_S} ((T \setminus U) \sqcap (T \sqcap U)) \approx \emptyset$$
  $$\models_{\mathfrak{T}_S} ((U \setminus T) \sqcap (T \sqcap U)) \approx \emptyset$$

  Property 6 follows from:

  $$\models_{\mathfrak{T}_S} T \approx ((T \setminus U) \sqcup (T \sqcap U))$$
  $$\models_{\mathfrak{T}_S} U \approx ((U \setminus T) \sqcup (T \sqcap U))$$
  $$\models_{\mathfrak{T}_S} (T \sqcap U) \approx (T \sqcap U)$$
  $$\models_{\mathfrak{T}_S} (U \setminus T) \approx (U \setminus T)$$
  $$\models_{\mathfrak{T}_S} (T \setminus U) \approx (T \setminus U)$$

  and reasoning as earlier that any constraint of the form $s' \approx \emptyset$ does not affect the property.
- s is $T \setminus U$ or $U \setminus T$: reasoning same as for $T \sqcap U$.

- s is $T \sqcup U$. If not already present, T, U, $T \setminus U$, $T \sqcap U$ are added to the graph as for $T \sqcap U$. In addition, add for union also adds $T \sqcup U$, and three edges. The properties follows from the following tautologies in $\mathfrak{T}_S$ in addition to those listed in analysis for $T \sqcap U$:

$$\models_{\mathfrak{T}_S} ((T \setminus U) \sqcap ((U \setminus T))) \approx \emptyset$$
$$\models_{\mathfrak{T}_S} (T \sqcup U) \approx ((T \setminus U) \sqcup (T \sqcap U) \sqcup (U \setminus T))$$

Finally, we show that if $\mathcal{G}$ satisfies the properties, then so does merge$(\mathcal{G}, s, t)$ if $s \in V(\mathcal{G})$, $t \in V(\mathcal{G})$, $\mathcal{L}(s) \nsubseteq \mathcal{L}(t)$ and $\mathcal{L}(t) \nsubseteq \mathcal{L}(s)$.

Let L_s denote $\mathcal{L}(s)$ in $\mathcal{G}$, and L'_s denote $\mathcal{L}(s)$ in merge$(\mathcal{G}, s', t')$ (likewise for t, u etc.).

In order to show property 5 holds, let $s' \in V(\mathcal{G})$, $t' \in L'_{s'}$ and $u' \in L'_{s'}$. We need to show: $\models_{\mathfrak{T}_S} t' \sqcap u' \approx \emptyset$.

- Let $t' \in L_{s'}$ and $u' \in L_{s'}$, i.e. both are also leaf nodes in $\mathcal{G}$. Then, the property for merge$(\mathcal{G}, s, t)$ follows from that of $\mathcal{G}$.
- Let t' be one of the newly introduced leaf nodes and $u' \in L_{s'}$ a leaf node in $\mathcal{G}$. Without loss of generality, let t' be $t_1 \sqcap t_2$ with $t_1 \in L_s \setminus L_t$ and $t_2 \in L_t \setminus L_s$. For t' to be in $L'_{s'}$, given the way the edges are added, either $t_1 \in L_{s'}$ or $t_2 \in L_{s'}$. Thus, we know that either $\models_{\mathfrak{T}_S} t_1 \sqcap u' \approx \emptyset$ or $\models_{\mathfrak{T}_S} t_2 \sqcap u' \approx \emptyset$. In either case, it follows that $\models_{\mathfrak{T}_S} (t_1 \sqcap t_2) \sqcap u' \approx \emptyset$, i.e. $\models_{\mathfrak{T}_S} t' \sqcap u' \approx \emptyset$.
- The analysis for the case where both are newly introduced leaf nodes is similar.

To show property 6 holds, the main observation is that each node no longer a leaf node, say $s' \in L_s \setminus L'_s$, is union of a new set of leaf nodes in L'_s (assuming the equalities).

$$\begin{aligned}
s' &\approx s' \sqcap s && (s' \in L_s,\ s \approx \bigsqcup_{s'' \in L_s} s'') \\
&\approx s' \sqcap t && (s \approx t \in E) \\
&\approx s' \sqcap \left(\bigsqcup_{t' \in L_t} t' \right) && (t \approx \bigsqcup_{t' \in L_t} t') \\
&\approx \bigsqcup_{t' \in L_t} s' \sqcap t' && (\text{distribute})
\end{aligned}$$

But by property 5, $s' \sqcap t' \approx \emptyset$ for $s', t' \in L_s$. Thus,

$$s' \approx \bigsqcup_{t' \in L_t \setminus L_s} s' \sqcap t'$$

Note that $\{s' \sqcap t' \mid t' \in L_t \setminus L_s\}$ are precisely the nodes in L'_s to which edges are added from s'. The proof for a node in L_t but not in L'_t is similar.

Since all the new leaf nodes are of the form $s' \sqcap t'$ with $s' \in L_s \setminus L_t$ and $t' \in L_t \setminus L_s$, it follows that property 6 holds for merge$(\mathcal{G}, s, t)$ if it holds for $\mathcal{G}$ assuming $s \approx t \in E$. □

Proposition 4.5. *Let $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G} \rangle$ be a state such that none of the rules in our calculus are applicable. Let $\mathfrak{S}$ be an interpretation defined in Proposition 4.3 satisfying constraints in $\mathcal{S}$ and $\mathcal{M}$. To recall, for x and y of* Element *sort,*

$$x^{\mathfrak{S}} = y^{\mathfrak{S}} \text{ if and only if } x \approx y \in \mathcal{M}^*$$

and for s of Set *sort,*

$$s^{\mathfrak{S}} = \left\{ x^{\mathfrak{S}} \;\middle|\; x \sqsubseteq s \in \mathcal{S}^* \right\}.$$

Let $\mathfrak{A}$ be an interpretation satisfying $\mathcal{A}$. Then, for all $t \in \mathcal{L}(\mathcal{G})$,

$$c_t^{\mathfrak{A}} \geq \left| t^{\mathfrak{S}} \right|.$$

Proof. Let $t \in \mathcal{L}(\mathcal{G})$. First we show that if $\mathcal{A} \Rightarrow c_t \geq |t_{\mathcal{S}}|$, then the proposition follows. That is there exists $n \geq |t_{\mathcal{S}}|$ such that c_t >= $n \in \mathcal{A}$. Let Elements$(\cdot)$ be as in (4.1).

$$\begin{aligned}
c_t^{\mathfrak{A}} &\geq n^{\mathfrak{A}} && (c_t \texttt{>=} n \in \mathcal{A}) \\
&= n && \text{(constant symbol)} \\
&\geq |t_{\mathcal{S}}| && \text{(definition)} \\
&= |\text{Elements}(t)| && (x^{\mathfrak{S}} = y^{\mathfrak{S}} \text{ iff } x \approx y \in \mathcal{M}^*) \\
&= \left| t^{\mathfrak{S}} \right| && \text{(using (4.2))}
\end{aligned}$$

It remains to show that $\mathcal{A} \Rightarrow c_t \geq |t_{\mathcal{S}}|$. Because of rule Members Arrangement, either $\mathcal{A} \Rightarrow c_t \geq |t_{\mathcal{S}}|$ or Rule Members Arrangement is applicable until the premise of rule Propagate Minsize holds. If Rule Propagate Minsize is applicable, c_t >= $|t_{\mathcal{S}}|$ must have been added to $\mathcal{A}$. In either case, $\mathcal{A} \Rightarrow c_t \geq |t_{\mathcal{S}}|$. □

Completeness is a direct consequence of the following result.

Proposition 4.6. *Let $\mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0$ be set, element and cardinality constraints respectively, satisfying Restriction 3.1. Let $\mathbf{D}$ be a derivation with respect to rules $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$ from state $\langle \mathcal{S}_0, \mathcal{M}_0, \mathcal{A}_0, (\emptyset, \emptyset) \rangle$. If $\mathbf{D}$ is finite, and the final derivation tree, say $\mathcal{D}$, in $\mathbf{D}$ is open and saturated with respect to the rules $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$; then there exists an interpretation $\mathfrak{I}$ that satisfies $\mathcal{S}_0$, $\mathcal{M}_0$ and $\mathcal{A}_0$.*

Proof. Proof outline: We build the model of the leaf nodes in the graph by modifying as needed the model obtained from Proposition 4.3. We add additional elements to these sets to make the cardinalities match the model satisfying the cardinality constraints and the constraints induced by the graph. Propositions 4.4 and 4.5 ensure that it is always possible to do so without violating the set constraints.

As $\mathcal{D}$ is open, there exists a branch that does not end in the state unsat. Let $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{G} \rangle$ be the final state on such a branch.

Let $\mathcal{A} \cup \hat{\mathcal{G}}$ be the cardinality constraints, and the cardinality constraints induced by the graph. These constraints fall in the theory $\mathfrak{T}_A$. Let $\mathfrak{A}$ be the structure satisfying these constraints. Such a structure exists because rule Arithmetic contradiction would have closed the branch if the constraints were inconsistent. From Proposition 4.3, we obtain an structure $\mathfrak{S}$ satisfying $\mathcal{S}$ and $\mathcal{M}$. Without loss of generality, assume that $\mathsf{Element}^{\mathfrak{S}}$ is infinite.

The $\mathfrak{I}$ we build satisfying $\mathcal{S}_0 \cup \mathcal{M}_0 \cup \mathcal{A}_0$ will be as follows. It coincides with the structure $\mathfrak{S}$ on terms of Element sort. It coincides with the structure $\mathfrak{A}$ on terms of Card sort. In order to define the value of set variables, for each leaf node $t \in$ Leaves$(\mathcal{G})$ we create the following sets:

$$B_t = \{ e_{t,1}, e_{t,2} \ldots e_{t, c_t^{\mathfrak{I}} - |t^{\mathfrak{S}}|} \}$$

where $e_{t,i} \in \mathsf{Element}^{\mathfrak{S}}$ are distinct from each other and from any e such that $e = x^{\mathfrak{S}}$ for x in $\mathcal{S}$ or $\mathcal{M}$. From Proposition 4.5, we know that $c_t^{\mathfrak{I}} \geq \left|t^{\mathfrak{S}}\right|$. Thus, for a leaf node t,

$$\left|t^{\mathfrak{S}}\right| + |B_t| = c_t^{\mathfrak{I}}. \tag{4.4}$$

For a set variable not in the graph, $S \notin V(\mathcal{G})$, define $S^{\mathfrak{I}} = S^{\mathfrak{S}}$. For a set variable in the graph, $S \in V(\mathcal{G})$, define:

$$S^{\mathfrak{I}} = \bigcup_{t \in \mathcal{L}(S)} (t^{\mathfrak{S}} \cup B_t) \tag{4.5}$$

From Proposition 4.4, it follows that:

$$\bigcup_{t \in \mathcal{L}(S)} t^{\mathfrak{S}} = S^{\mathfrak{S}} \tag{4.6}$$

So an equivalent way to define $S^{\mathfrak{I}}$ is as follows:

$$S^{\mathfrak{I}} = S^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(S)} B_t \tag{4.7}$$

We verify that each constraints in $\mathcal{S}_0$ is satisfied:

(1) $S \approx T$, $S \not\approx T$.

For $S \approx T$, we need to show $S^{\mathfrak{I}} = T^{\mathfrak{I}}$. If neither $S \in V(\mathcal{G})$ nor $T \in V(\mathcal{G})$, then this follows from Proposition 4.3. If either $S \in V(\mathcal{G})$ or $T \in V(\mathcal{G})$, then due to rule Introduce Eq Right and rule Introduce Eq Left both $S \in V(\mathcal{G})$ and $T \in V(\mathcal{G})$. From Proposition 4.4, property 1, we know that $\mathcal{L}(S) = \mathcal{L}(T)$. From the definition of $S^{\mathfrak{I}}$ and $T^{\mathfrak{I}}$ in (4.5), it follows that $S^{\mathfrak{I}} = T^{\mathfrak{I}}$.

For $S \not\approx T$, we need to show $S^{\mathfrak{I}} \neq T^{\mathfrak{I}}$. Let us write $S^{\mathfrak{I}} = S^{\mathfrak{S}} \cup B_S$, where $B_S = \emptyset$ if $S \notin V(\mathcal{G})$, otherwise let $B_S = \bigcup_{t \in \mathcal{L}(S)} B_t$ (from (4.7)). Similarly we may write $T^{\mathfrak{I}} = T^{\mathfrak{S}} \cup B_T$. From Proposition 4.3 we know that $S^{\mathfrak{S}} \neq T^{\mathfrak{S}}$. Without loss of generality assume $e \in S^{\mathfrak{S}}$ and $e \notin T^{\mathfrak{S}}$. By definition, B_T is disjoint from $S^{\mathfrak{S}}$, thus $e \notin B_T$. Thus, $e \in S^{\mathfrak{I}}$ and $e \notin T^{\mathfrak{I}}$. $S^{\mathfrak{I}} \neq T^{\mathfrak{I}}$ follows.

(2) $S \approx \emptyset$.

We need to show $S^{\mathfrak{I}} = \emptyset^{\mathfrak{I}} = \emptyset$. It will follow from rule Introduce Empty Set and rule Introduce Eq Left.

$\emptyset \in V(\mathcal{G})$ and $S \in V(\mathcal{G})$	(Rules Introduce Empty Set, Introduce Eq Left)
$\mathcal{L}(S) = \mathcal{L}(\emptyset)$	(Proposition 4.4, property 1)
$\mathcal{L}(S) = \emptyset$	$(\mathcal{L}(\emptyset) = \emptyset)$
$S^{\mathfrak{I}} = \emptyset$	$(S \in V(\mathcal{G})$, (4.5))

(3) $S \approx \{x\}$.

We need to show that $S^{\mathfrak{I}} = \left\{x^{\mathfrak{I}}\right\}$. From rule Introduce Singleton we conclude that $\{x\} \in V(\mathcal{G})$ Then, from rule Introduce Eq Left, $S \in V(\mathcal{G})$.

From $\hat{\mathcal{G}}$, we know that:

$$\begin{aligned} c_S^{\mathfrak{I}} &= \sum_{t \in \mathcal{L}(S)} c_t^{\mathfrak{I}} && \text{(constraint in } \hat{\mathcal{G}} \text{ for } c_S) \\ &= \sum_{t \in \mathcal{L}(\{x\})} c_t^{\mathfrak{I}} && \text{(Proposition 4.4, property 1)} \\ &= c_{\{x\}}^{\mathfrak{I}} && \text{(constraint in } \hat{\mathcal{G}} \text{ for } c_{\{x\}}) \\ &= 1 && \text{(constraint in } \hat{\mathcal{G}} \text{ for singletons)} \end{aligned}$$

We can conclude that $\left|S^{\mathfrak{I}}\right| = 1$ as $\left|S^{\mathfrak{I}}\right| = c_S^{\mathfrak{I}}$ (for proof of $\left|S^{\mathfrak{I}}\right| = c_S^{\mathfrak{I}}$, see reasoning later in this proof for $|S| \approx c_S$ – the same reasoning works for all nodes $S \in V(\mathcal{G})$)

From, Singleton, we know $x^{\mathfrak{S}} \in S^{\mathfrak{S}}$. By Proposition 4.3, $x^{\mathfrak{S}} \in S^{\mathfrak{S}}$. As

$$S^{\mathfrak{I}} = S^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(S)} B_t$$

and $\left|S^{\mathfrak{I}}\right| = 1$, we conclude that $S^{\mathfrak{I}} = \left\{x^{\mathfrak{S}}\right\} = \left\{x^{\mathfrak{I}}\right\}$.

(4) $S \approx T \sqcup U$. We need to show $S^{\mathfrak{I}} = T^{\mathfrak{I}} \cup U^{\mathfrak{I}}$.

Let $S \notin V(\mathcal{G})$, $T \notin V(\mathcal{G})$, and $U \notin V(\mathcal{G})$. Then,

$$\begin{aligned} S^{\mathfrak{I}} &= S^{\mathfrak{S}} && (S \notin V(\mathcal{G})) \\ &= T^{\mathfrak{S}} \cup U^{\mathfrak{S}} && \text{(Proposition 4.3)} \\ &= T^{\mathfrak{I}} \cup U^{\mathfrak{I}} && (T \notin V(\mathcal{G}),\, U \notin V(\mathcal{G})) \end{aligned}$$

Otherwise, let $S \in V(\mathcal{G})$, or $T \in V(\mathcal{G})$, or $U \in V(\mathcal{G})$. Then, from Rules Introduce Eq Right, Introduce Eq Left, Introduce Union and definition of add, we know S, T, *and* U in $V(\mathcal{G})$. Then,

$$\begin{aligned} S^{\mathfrak{I}} &= \bigcup_{t \in \mathcal{L}(S)} (t^{\mathfrak{S}} \cup B_t) && (S \in V(\mathcal{G})) \\ &= \bigcup_{t \in \mathcal{L}(T \sqcup U)} (t^{\mathfrak{S}} \cup B_t) && \text{(Proposition 4.4)} \\ &= \left(\bigcup_{t \in \mathcal{L}(T)} (t^{\mathfrak{S}} \cup B_t) \right) \cup \left(\bigcup_{t \in \mathcal{L}(U)} (t^{\mathfrak{S}} \cup B_t) \right) && \text{(Proposition 4.4)} \\ &= T^{\mathfrak{I}} \cup U^{\mathfrak{I}} && (T \in V(\mathcal{G}),\, U \in V(\mathcal{G})) \end{aligned}$$

(5) $S \approx T \sqcap U$. We need to show $S^{\mathfrak{I}} = T^{\mathfrak{I}} \cap U^{\mathfrak{I}}$.

Let $S \notin V(\mathcal{G})$, $T \notin V(\mathcal{G})$, and $U \notin V(\mathcal{G})$. Then,

$$\begin{aligned} S^{\mathfrak{I}} &= S^{\mathfrak{S}} && (S \notin V(\mathcal{G})) \\ &= T^{\mathfrak{S}} \cap U^{\mathfrak{S}} && \text{(Proposition 4.3)} \\ &= T^{\mathfrak{I}} \cap U^{\mathfrak{I}} && (T \notin V(\mathcal{G}),\, U \notin V(\mathcal{G})) \end{aligned}$$

Let $S \notin V(\mathcal{G})$ and $T \notin V(\mathcal{G})$, but $U \in V(\mathcal{G})$. Then,

$$\begin{aligned}
T^{\mathfrak{I}} \cap U^{\mathfrak{I}} &= T^{\mathfrak{S}} \cap U^{\mathfrak{I}} && (T \notin V(\mathcal{G})) \\
&= T^{\mathfrak{S}} \cap \left(U^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(U)} B_t \right) && (U \in V(\mathcal{G})) \\
&= T^{\mathfrak{S}} \cap U^{\mathfrak{S}} && (T^{\mathfrak{S}} \cap B_t = \emptyset) \\
&= S^{\mathfrak{S}} && \text{(Proposition 4.3)} \\
&= S^{\mathfrak{I}} && (S \notin V(\mathcal{G}))
\end{aligned}$$

If $S \notin V(\mathcal{G})$ and $U \notin V(\mathcal{G})$, but $T \in V(\mathcal{G})$; the reasoning is same as above.

Otherwise, either $S \in V(\mathcal{G})$ or both $T \in V(\mathcal{G})$ and $U \in V(\mathcal{G})$. Then, from Rules Introduce Eq Right, Introduce Eq Left, Introduce Inter and definition of add, we know S, T, *and* U in $V(\mathcal{G})$. Then,

$$T^{\mathfrak{I}} \cap U^{\mathfrak{I}} = \left(T^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(T)} B_t \right) \cap \left(U^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(U)} B_t \right) \qquad (T, U \text{ in } V(\mathcal{G}))$$

As each B_t is disjoint from all other sets, the above expression simplifies to:

$$\begin{aligned}
&= \left(T^{\mathfrak{S}} \cap U^{\mathfrak{S}} \right) \cup \bigcup_{t \in \mathcal{L}(T) \cap \mathcal{L}(U)} B_t \\
&= S^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(S)} B_t && \text{(Propositions 4.3 and 4.4)} \\
&= S^{\mathfrak{I}} && (S \in V(\mathcal{G}))
\end{aligned}$$

(6) $S \approx T \setminus U$. We need to show $S^{\mathfrak{I}} = T^{\mathfrak{I}} \setminus U^{\mathfrak{I}}$.

Let $S \notin V(\mathcal{G})$, $T \notin V(\mathcal{G})$, and $U \notin V(\mathcal{G})$. Then,

$$\begin{aligned}
S^{\mathfrak{I}} &= S^{\mathfrak{S}} && (S \notin V(\mathcal{G})) \\
&= T^{\mathfrak{S}} \setminus U^{\mathfrak{S}} && \text{(Proposition 4.3)} \\
&= T^{\mathfrak{I}} \setminus U^{\mathfrak{I}} && (T \notin V(\mathcal{G}),\, U \notin V(\mathcal{G}))
\end{aligned}$$

Let $S \notin V(\mathcal{G})$ and $T \notin V(\mathcal{G})$, but $U \in V(\mathcal{G})$. Then,

$$\begin{aligned}
T^{\mathfrak{I}} \setminus U^{\mathfrak{I}} &= T^{\mathfrak{S}} \setminus U^{\mathfrak{I}} && (T \notin V(\mathcal{G})) \\
&= T^{\mathfrak{S}} \setminus \left(U^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(U)} B_t \right) && (U \in V(\mathcal{G})) \\
&= T^{\mathfrak{S}} \setminus U^{\mathfrak{S}} && (T^{\mathfrak{S}} \setminus B_t = T^{\mathfrak{S}}) \\
&= S^{\mathfrak{S}} && \text{(Proposition 4.3)} \\
&= S^{\mathfrak{I}} && (S \notin V(\mathcal{G}))
\end{aligned}$$

Note that in contrast to intersection, if $S \notin V(\mathcal{G})$, $T \in V(\mathcal{G})$, and $U \notin V(\mathcal{G})$, the above analysis does not apply. We do need to introduce and reason about the equality in the graph.

Let $S \in V(\mathcal{G})$ or $T \in V(\mathcal{G})$. From Rules Introduce Eq Right, Introduce Eq Left, Introduce Set difference and definition of add we know S, T, *and* U in $V(\mathcal{G})$. Then,

$$T^{\mathfrak{I}} \setminus U^{\mathfrak{I}} = \left(T^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(T)} B_t\right) \setminus \left(U^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(U)} B_t\right) \qquad (T, U \text{ in } V(\mathcal{G}))$$

As each B_t is disjoint from all other sets, the above expression simplifies to:

$$\begin{aligned} &= \left(T^{\mathfrak{S}} \setminus U^{\mathfrak{S}}\right) \cup \bigcup_{t \in \mathcal{L}(T) \setminus \mathcal{L}(U)} B_t \\ &= S^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(S)} B_t && \text{(Propositions 4.3 and 4.4)} \\ &= S^{\mathfrak{I}} && (S \notin V(\mathcal{G})) \end{aligned}$$

(7) $x \sqsubseteq S$, $x \not\sqsubseteq S$.

Note that irrespective of whether $S \in V(\mathcal{G})$ or $S \notin V(\mathcal{G})$, $S^{\mathfrak{S}} \subseteq S^{\mathfrak{I}}$. Thus, from Proposition 4.3, $x^{\mathfrak{I}} \in S^{\mathfrak{I}}$ if $x \sqsubseteq S$ is a constraint.

It remains to show that if $x \not\sqsubseteq S$ is a constraint then $x^{\mathfrak{I}} \notin S^{\mathfrak{I}}$. If $S \notin V(\mathcal{G})$, then again $x^{\mathfrak{I}} \notin S^{\mathfrak{I}}$ follows from Proposition 4.3. If $S \in V(\mathcal{G})$, then observe that $S^{\mathfrak{I}}$ is $S^{\mathfrak{S}} \cup \bigcup_{t \in \mathcal{L}(U)} B_t$. We already know $x^{\mathfrak{I}} \notin S^{\mathfrak{S}}$. It remains to show that $x^{\mathfrak{I}} \notin \bigcup_{t \in \mathcal{L}(U)} B_t$. This follows from the definition of B_t.

(8) $c_S \approx \mathsf{card}(S)$.

From Proposition 4.4, we know that for t, u in $\mathcal{L}(S)$:

$$\models_{\mathfrak{T}_S} t \sqcap u \approx \emptyset$$

and also,

$$\models_{\mathfrak{T}_S} \left(\bigwedge_{t \in E} t \approx \emptyset\right) \Rightarrow \left(S \approx \bigsqcup_{t \in \mathcal{L}(S)} t\right)$$

where $E = \{t \in V(\mathcal{G}) \mid t \approx \emptyset \in \mathcal{S}^*\}$.

In $\mathfrak{I}$, as for each $t \in E$, $t^{\mathfrak{I}} = \emptyset$, it follows that:

$$S^{\mathfrak{I}} = \bigcup_{t \in \mathcal{L}(S)} t^{\mathfrak{I}} \quad .$$

Also, for t, u in $\mathcal{L}(S)$:

$$t^{\mathfrak{I}} \cap u^{\mathfrak{I}} = \emptyset \quad .$$

In other words, $S^{\mathfrak{I}}$ is a disjoint union of $t^{\mathfrak{I}}$ where $t \in \mathcal{L}(S)$. It follows that,

$$\left|S^{\mathfrak{I}}\right| = \sum_{t \in \mathcal{L}(S)} \left|t^{\mathfrak{I}}\right|$$

For a leaf node $t \in \mathcal{L}(S)$, from (4.4) we know that $\left|t^{\mathfrak{I}}\right| = \left|t^{\mathfrak{S}}\right| + |B_t| = c_t^{\mathfrak{I}}$. We may thus conclude,

$$\left|S^{\mathfrak{I}}\right| = \sum_{t \in \mathcal{L}(S)} c_t^{\mathfrak{I}}$$

From the constraint on cardinality for S induced by the graph, i.e the constraint on c_S in $\hat{\mathcal{G}}$, we know that $c_S^{\mathfrak{I}} = \sum_{t \in \mathcal{L}(S)} c_t^{\mathfrak{I}}$. The result follows:

$$\left| S^{\mathfrak{I}} \right| = c_S^{\mathfrak{I}}$$

□

Proposition 4.7 (Completeness)**.** *Under any fair derivation strategy, every derivation of a set $\mathcal{C}$ of $\mathfrak{T}_S$-unsatisfiable constraints extends to a refutation.*

Proof. Contrapositively, suppose that $\mathcal{C}$ has a derivation **D** that cannot be extended to a refutation. By Proposition 4.2, **D** must be extensible to one that ends with a tree with a saturated branch. By Proposition 4.6, $\mathcal{C}$ is satisfiable in $\mathfrak{T}_S$. □

4.3. **Soundness.** We start by showing that every rule preserves constraint satisfiability.

Lemma 4.8. *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.*

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 4.4, 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 4.4 (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 4.9 (Soundness)**.** *Every set of $\mathfrak{T}_S$-constraints that has a refutation is $\mathfrak{T}_S$-unsatisfiable.*

Sketch. Given Lemma 4.8, 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 $\mathcal{C}$ of $\mathfrak{T}_S$-constraints starts with a state $\mathfrak{T}_S$-equisatisfiable with $\mathcal{C}$. □

5. Evaluation

We have implemented a decision procedure based on the calculus above in the SMT solver cvc4 [BCD+11]. We describe a high-level, non-deterministic version of it here, followed by an experimental 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 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 of the derivation tree being constructed. Initially, the current branch is the only branch in the 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.
(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 a software verification applications. 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.[7] Benchmarks are available on the cvc4 website.[8]

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 [KNR06, KR07, SSK11]. The results from running cvc4 on these benchmarks are provided in top half of Table 1. 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[9] (# L) in the graph at any point during the run of the algorithm. Keeping the number of leaves low is important to avoid a blowup from the Merge Equality II rule.

Although we have not rerun the systems described in [KNR06, KR07, SSK11], we report here the experimental results as stated in the respective papers.[10] Since the experiments

[7] `https://github.com/kbansal/CVC4/tree/37f6117`

[8] `http://cvc4.cs.stanford.edu/papers/LMCS-2018/`

[9] 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.

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

TABLE 1. Performance of our calculus on benchmarks derived from verification of programs

file	**output**	**time (s.)**	**# vertices**	**# leaves**
cade07-vc1.smt2	unsat	0.00	3	3
cade07-vc2a.smt2	unsat	0.00	6	3
cade07-vc2b.smt2	sat	0.01	15	5
cade07-vc2.smt2	unsat	0.01	6	3
cade07-vc3a.smt2	unsat	0.00	6	0
cade07-vc3b.smt2	sat	0.02	15	6
cade07-vc3.smt2	unsat	0.01	6	0
cade07-vc4b.smt2	sat	0.16	44	12
cade07-vc4.smt2	unsat	0.17	51	16
cade07-vc5b.smt2	sat	0.39	63	21
cade07-vc5.smt2	unsat	0.38	77	25
cade07-vc6a.smt2	unsat	0.02	32	12
cade07-vc6b.smt2	sat	0.04	32	12
cade07-vc6c.smt2	sat	0.06	32	12
cade07-vc6.smt2	unsat	0.32	36	16
cvc4-card.scala-10.smt2	2 sat/2 unsat	0.10	48	19
cvc4-card.scala-12.smt2	1 sat/3 unsat	0.03	0	0
cvc4-card.scala-14.smt2	2 sat/2 unsat	0.09	25	11
cvc4-card.scala-15.smt2	1 sat/3 unsat	0.01	0	0
cvc4-card.scala-16.smt2	2 sat/4 unsat	0.26	39	18
cvc4-card.scala-17.smt2	1 sat/3 unsat	0.02	19	8
cvc4-card.scala-18.smt2	2 sat/2 unsat	0.10	39	20
cvc4-card.scala-21.smt2	2 sat/2 unsat	1.69	134	35
cvc4-card.scala-6.smt2	1 sat/4 unsat	0.02	8	5
cvc4-card.scala-8.smt2	1 sat/3 unsat	0.06	21	12

were run on different machines the comparison is only indicative, but it does suggest that our solver has comparable performance.

In [KR07], the procedure from [KNR06] is reported to solve 12 of the 15 benchmarks with a timeout of 100 seconds, while the novel procedure in [KR07] is reported to solve 11 of the 15 benchmarks with the same timeout. The best-performing previous procedure ([SSK11]) can solve all 15 benchmarks in under a second.[11] As another point of comparison, we tested the procedure from [SSK11] 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 solver 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.

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 [BKKS13]. These contain a mix of membership and cardinality

[11] [SSK11] includes a second set of benchmarks, but we were unable to evaluate our procedure on these, as they were only made available in a non-standard format and were missing crucial datatype declarations.

constraints combined with constraints over the theories of datatypes and bitvectors. The results of this evaluation are shown in bottom half of Table 1. 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 SMT solver CVC4, and demonstrated the feasibility as well as some advantages of our approach. We hope this work will enable the use of sets and cardinality constraints in many new applications that rely on SMT solvers. We also expect to use it to drive the development of a standard theory of sets under the SMT-LIB initiative [BFT].

We expect to pursue several directions of future work. We will investigate relaxing Restriction 3.1 by doing more reasoning modulo equality. We will also experiment with different strategies to attempt to find the most efficient ones. We will also look into efficient means of combining sets with other theories and investigate extensions to relations and relational operators.

Acknowledgement

The authors wish to acknowledge fruitful discussions with Viktor Kuncak and Etienne Kneuss and for providing the Leon benchmarks. We thank Philippe Suter for his help running the algorithm from [SSK11].

References

[AA05] Jean-Raymond Abrial and Jean-Raymond Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, 2005.

[AGP16] Francesco Alberti, Silvio Ghilardi, and Elena Pagani. Counting constraints in flat array fragments. In *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, pages 65–81, 2016.

[ASM80] Jean-Raymond Abrial, Stephen A. Schuman, and Bertrand Meyer. Specification language. In *On the Construction of Programs*, pages 343–410. Cambridge University Press, 1980.

[Ban16] Kshitij Bansal. *Decision Procedures for Finite Sets with Cardinality and Local Theory Extensions.* PhD thesis, New York University, January 2016.

[BCD+11] 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.

[BFT] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `http://www.SMT-LIB.org`.

[BKKS13] 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.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[BRBT16] Kshitij Bansal, Andrew Reynolds, Clark Barrett, and Cesare Tinelli. A new decision procedure for finite sets and cardinality constraints in SMT. In Nicola Olivetti and Ashish Tiwari, editors, *Proceedings of the* 8th *International Joint Conference on Automated Reasoning (IJCAR '16)*, volume 9706 of *Lecture Notes in Computer Science*, pages 82–98. Springer International Publishing, June 2016. Coimbra, Portugal.

[BS17] Markus Bender and Viorica Sofronie-Stokkermans. Decision procedures for theories of sets with measures. In *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, pages 166–184, 2017.

[BSST09] 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.

[CFR15] Paula Chocron, Pascal Fontaine, and Christophe Ringeissen. A polite non-disjoint combination method: Theories with bridging functions revisited. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, pages 419–433, 2015.

[COP01] Domenico Cantone, Eugenio Omodeo, and Alberto Policriti. *Set Theory for Computing. From Decision Procedures to Logic Programming with Sets*. Monographs in Computer Science. Springer, 2001.

[CR16] Maximiliano Cristiá and Gianfranco Rossi. A decision procedure for sets, binary relations and partial functions. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, pages 179–198, 2016.

[CZ98] 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.

[DMB09] 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.

[Jac12] Daniel Jackson. *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[JB10] Dejan Jovanović and Clark Barrett. Polite theories revisited. In *Proceedings of the* 17th *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '10)*, volume 6397 of *LNCS*, pages 402–416. Springer, October 2010.

[KNR06] Viktor Kuncak, HuuHai Nguyen, and Martin Rinard. Deciding Boolean algebra with Presburger arithmetic. *Journal of Automated Reasoning*, 36(3):213–239, 2006.

[KR07] 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.

[KRW09] 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* $7^{t}h$ *International Workshop on Satisfiability Modulo Theories (SMT '09)*, August 2009.

[NOT06] 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.

[PK08] Ruzica Piskac and Viktor Kuncak. Decision procedures for multisets with cardinality constraints. In *Verification, Model Checking, and Abstract Interpretation, 9th International Conference, VMCAI 2008, San Francisco, USA, January 7-9, 2008, Proceedings*, pages 218–232, 2008.

[SDSD86] Jacob T. Schwartz, Robert B. K. Dewar, Edmond Schonberg, and Ed Dubinsky. *Programming with Sets; an Introduction to SETL*. Springer-Verlag New York, Inc., New York, NY, USA, 1986.

[SS09] Viorica Sofronie-Stokkermans. Locality results for certain extensions of theories with bridging functions. In *Proceedings of the 22Nd International Conference on Automated Deduction*, CADE-22, pages 67–83, Berlin, Heidelberg, 2009. Springer-Verlag.

[SSK11] Philippe Suter, Robin Steiger, and Viktor Kuncak. Sets with cardinality constraints in Satisfiability Modulo Theories. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2011.

[vGBR16] Klaus von Gleissenthall, Nikolaj Bjørner, and Andrey Rybalchenko. Cardinalities and universal quantifiers for verifying parameterized systems. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 599–613, 2016.

[Zar02] Calogero G. Zarba. Combining sets with integers. In *Frontiers of Combining Systems, 4th International Workshop, FroCoS 2002*, pages 103–116, 2002.

[Zar05] Calogero G. Zarba. Combining sets with cardinals. *Journal of Automated Reasoning*, 34(1):1–29, Jan 2005.