Sets with Cardinality Constraints in Satisfiability Modulo Theories

Philippe Suter*, Robin Steiger, and Viktor Kuncak

École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
firstname.lastname@epfl.ch

Abstract. Boolean Algebra with Presburger Arithmetic (BAPA) is a decidable logic that can express constraints on sets of elements and their cardinalities. Problems from verification of complex properties of software often contain fragments that belong to quantifier-free BAPA (QFBAPA). In contrast to many other NP-complete problems (such as quantifier-free first-order logic or linear arithmetic), the applications of QFBAPA to a broader set of problems has so far been hindered by the lack of an efficient implementation that can be used alongside other efficient decision procedures. We overcome these limitations by extending the efficient SMT solver Z3 with the ability to reason about cardinality (QFBAPA) constraints. Our implementation uses the DPLL(T) mechanism of Z3 to reason about the top-level propositional structure of a QFBAPA formula, improving the efficiency compared to previous implementations. Moreover, we present a new algorithm for automatically decomposing QFBAPA formulas. Our algorithm alleviates the exponential explosion of considering all Venn regions, significantly improving the tractability of formulas with many set variables. Because it is implemented as a theory plugin, our implementation enables Z3 to prove formulas that use QFBAPA constructs with constructs from other theories that Z3 supports, as well as with quantifiers. We have applied our implementation to the verification of functional programs; we show it can automatically prove formulas that no automated approach was reported to be able to prove before.

1 Introduction

Sets naturally arise in software that performs discrete computation, as a built-in data type [2], as container libraries, or inside program specification constructs [11, 20]. An intrinsic part of reasoning about sets is reasoning about sizes of sets, with well-known associated laws such as the inclusion-exclusion principle $|A \cup B| = |A| + |B| - |A \cap B|$. A natural decidable logic that supports set operations (union, intersection, complement) as well as a cardinality operator is a logic we call BAPA, for Boolean Algebra with Presburger Arithmetic [3, 8].

We here consider the quantifier-free fragment of BAPA, denoted QFBAPA. QFBAPA was shown to be NP-complete using a particular encoding into

* Philippe Suter was supported by the Swiss NSF Grant 200021_120433.

quantifier-free Presburger arithmetic that exploits an integer analogue of Carathéodory theorem [10]. We thus think of QFBAPA as a generalization of SAT that is similar to SAT from a high-level complexity-theory point of view. The richness of QFBAPA is reflected in the fact that, being propositionally closed, it subsumes SAT. Moreover, unlike SAT, no encoding is needed to represent integer linear arithmetic in BAPA. Crucially, BAPA supports set operations and cardinality, whose polynomial encoding into SAT is possible but non-trivial [10]. Strikingly, a number of expressive logics can be naturally reduced to QFBAPA [17, 9]. This enables combination of formulas from non-disjoint theory signatures that share set operations, and goes beyond current disjoint theory combinations.

However, although the QFBAPA satisfiability problem is NP-complete, the resulting algorithm has proven difficult for unsatisfiable QFBAPA instances; its improvements have primarily helped dealing with satisfiable instances [10]. This empirical observation is in contrast to the situation for propositional logic, for which the community developed SAT solvers that are effective in showing unsatisfiability for large industrial benchmarks. In this paper we present a QFBAPA implementation that is effective both for satisfiable and unsatisfiable instances.

Our implementation incorporates an important new algorithmic component: a decomposition of constraints based on the hypergraphs of common set variables. This component analyzes the variables occurring in different atomic formulas within a QFBAPA formula and uses the structural property of the formula to avoid generating all Venn regions. Our implementation is integrated into the state-of-the-art SMT solver Z3, whose important feature is efficient support for linear arithmetic [13]. Efficient integration with Z3 was made possible by the recently introduced theory plugin architecture of Z3, as well as by an incremental implementation of our algorithm. In this integration, Z3 processes top level propositional structure of the formula, providing QFBAPA solver with conjunctions of QFBAPA constraints. Our solver generates lemmas in integer linear arithmetic and gives them back to Z3, which incorporates them with other integer constraints. At the same time, Z3 takes care of equality constraints. The net result is (1) dramatic improvement of efficiency compared to previously reported QFBAPA implementations (2) the ability to use QFBAPA cardinality operation alongside all other operations that Z3 supports. We illustrate the usefulness of this approach through experimental results that prove the validity of complex verifications condition arising from the verification of imperative as well as functional programs.

Contributions. In summary, our paper makes the following contributions:

- decomposition theorems and algorithms for efficient handling of QFBAPA cardinality constraints, often avoiding the need for exponentially many Venn regions as in [8], and avoiding complex conditional sums as in [10];
- an incremental algorithm to analyze the structure of QFBAPA formulas and generate the integer constraints following the decomposition theorems;
- an implementation of these algorithms as a theory plugin for the Z3 solver, with support for detecting equalities entailed by QFBAPA constraints and therefore precise combination with other theories supported by Z3;

– encouraging experimental results for benchmarks arising from verification of imperative and functional programs.

2 Example

We next illustrate the expressive power of the SMT prover we obtained by incorporating our QFBAPA decision procedure into Z3. Given a list datatype in a functional programming language, consider the question of proving that the set of elements contained in the list has a cardinality always less than or equal to the length of the list. The set of elements contained in the list and the length of the list are computed using natural recursive functions, specified below in the syntax of the Scala programming language:

```
def content(list: List[Int]) : Set[Int] = list match {
  case Nil ⇒ ∅
  case Cons(x, xs) ⇒ {x} ∪ content(xs)
}
def length(list: List[Int]) : Int = list match {
  case Nil ⇒ 0
  case Cons(x, xs) ⇒ 1 + length(xs)
}
```

Our goal is to prove the property:

$$\forall list : \mathsf{List[Int]} \;.\; |\,\mathsf{content}(list)\,| \leq \mathsf{length}(list)$$

We proceed by unfolding the recursive definitions sufficiently many times to obtain the following verification condition:

$$\begin{aligned} & list \neq \mathsf{Nil} \implies list = \mathsf{Cons}(x, xs) \\ \wedge\ & \mathsf{length}(\mathsf{Nil}) = 0 \wedge \mathsf{length}(\mathsf{Cons}(x, xs)) = 1 + \mathsf{length}(xs) \\ \wedge\ & \mathsf{content}(\mathsf{Nil}) = \emptyset \wedge \mathsf{content}(\mathsf{Cons}(x, xs)) = \{x\} \cup \mathsf{content}(xs) \\ \wedge\ & |\,\mathsf{content}(xs)\,| \leq \mathsf{length}(xs) \\ \implies & |\,\mathsf{content}(list)\,| \leq \mathsf{length}(list) \end{aligned}$$

Note that the formula includes reasoning about function symbols, integers, algebraic data types, sets, and cardinalities of sets. To the best of our knowledge, the implementation we present in this paper is the only one that is complete for proving the validity of formulas with such operators. The proof is found using Z3's DPLL(T) algorithm, which, among others, performs cases analysis on whether *list* is empty. It relies on Z3 to support arithmetic, congruence properties of functions, and algebraic data types. Finally, it crucially relies on invocations of our QFBAPA plugin. In response to currently asserted QFBAPA constraints, our plugin generates integer constraints on Venn regions that are chosen to ensure that the relationships between sets are handled in a complete way. The entire theorem proving process takes negligible time (see Section 5 for experimental results).

3 Decomposition in Solving BAPA Constraints

In this section, we consider formulas over finite sets of uninterpreted elements from a domain $\mathbb{E}$. We show in Section 4 how we combined QFBAPA with other theories to obtain a theory of sets of interpreted elements.

Syntax. Figure 1 presents the syntax of QFBAPA. We use $\mathsf{vars}(\phi)$ to denote the set of free set variables occurring in ϕ.

$$
\begin{array}{rcl}
\phi & ::= & A \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \\
A & ::= & S_1 = S_2 \mid S_1 \subseteq S_2 \mid T_1 = T_2 \mid T_1 \leq T_2 \\
S & ::= & s \mid \emptyset \mid \mathcal{U} \mid S_1 \cup S_2 \mid S_1 \cap S_2 \mid S_1 \setminus S_2 \mid S^{\mathsf{c}} \\
T & ::= & i \mid K \mid T_1 + T_2 \mid K \cdot T \mid \mid S \mid \\
K & ::= & \ldots \mid -2 \mid -1 \mid 0 \mid 1 \mid 2 \mid \ldots
\end{array}
$$

Fig. 1. Quantifier-Free Formulas of Boolean Algebra with Presburger Arithmetic (QFBAPA). S^{c} denotes the complement of the set S with respect to the universe $\mathcal{U}$, that is, $\mathcal{U} \setminus S$.

Definition 1 (Venn regions). *One central notion throughout the presentation of the theorems and the decision procedure is the notion of* Venn region*. A Venn region of n sets $S = \{S_1, \ldots, S_n\}$ is one of the 2^n set expressions described by $\bigcap_{i=1}^{n} S_i^{\alpha_i}$ where $S_i^{\alpha_i}$ is either S_i or S_i^{c}. By construction, the 2^n regions form a partition of $\mathcal{U}$. We write $\mathsf{venn}(S)$ to denote the set of all Venn regions formed with the sets in S.*

Semantics. An interpretation $\mathcal{M}$ of a QFBAPA formula ϕ is a map from the set variables of ϕ to finite subsets of $\mathbb{E}$ and from the integer variables of ϕ to values in $\mathbb{Z}$. It is a model of ϕ, denoted $\mathcal{M} \models \phi$ if the following conditions are satisfied:

- $\mathcal{U}^{\mathcal{M}}$ is a finite subset of $\mathbb{E}$ and $\emptyset^{\mathcal{M}}$ is the empty set
- for each set variable S of ϕ, $S^{\mathcal{M}} \subseteq \mathcal{U}^{\mathcal{M}}$
- for each integer variable k of ϕ, $k^{\mathcal{M}} \in \mathbb{Z}$
- when $=$, $\subseteq$, $\emptyset$, $\cup$, $\cap$, $\setminus$ and the cardinality function $|\cdot|$ are interpreted as expected, ϕ evaluates to true in $\mathcal{M}$

Definition 2. *We define $\sim_V$ to be the equivalence relation on interpretations, parametrized by a set of set variables V, such that:*

$$\mathcal{M}_1 \sim_V \mathcal{M}_2 \iff \forall v \in \mathsf{venn}(V) \,.\, |\, v^{\mathcal{M}_1} \,| = |\, v^{\mathcal{M}_2} \,|$$

Definition 3. *Let $\mathcal{M}$ be an interpretation and $f : \mathbb{E} \to \mathbb{E}$ a bijection from the interpretation domain to itself (a permutation function). We denote by $f[\mathcal{M}]$ the interpretation such that:*

- $\mathcal{U}^{f[\mathcal{M}]} = \{f(u) \mid u \in \mathcal{U}^{\mathcal{M}}\}$
- *for each set variable* S *interpreted in* $\mathcal{M}$, $S^{f[\mathcal{M}]} = \{f(u) \mid u \in S^{\mathcal{M}}\}$
- *for each integer variable* k *interpreted in* $\mathcal{M}$, $k^{f[\mathcal{M}]} = k^{\mathcal{M}}$

Theorem 1. *Let* ϕ *be a QFBAPA formula,* $\mathcal{M}$ *a model of* ϕ *and* $f : \mathbb{E} \to \mathbb{E}$ *a bijection, then* $f[\mathcal{M}] \models \phi$.

Proof. Prove by induction that $t^{f[\mathcal{M}]} = f[t^{\mathcal{M}}]$ for every set algebra term t. By bijectivity of f, $|\, t^{f[\mathcal{M}]} \,| = |\, f[t^{\mathcal{M}}] \,|$, so the values of all integer-valued terms remain invariant under f. Finally, note that $S_1 \subseteq S_2$ reduces to $|\, S_1 \setminus S_2 \,| = 0$ whereas $S_1 = S_2$ reduces to $|\, S_1 \setminus S_2 \,| = 0 \wedge |\, S_2 \setminus S_1 \,| = 0$.

3.1 A Simple Decision Procedure for QFBAPA

A simple technique for solving QFBAPA formulas is to reduce the problem to integer linear arithmetic as follows. Introduce a fresh integer variable for each Venn region in $\mathsf{venn}(\mathsf{vars}(\phi))$. Rewrite each constraint of the form $S_1 = S_2$ as $S_1 \subseteq S_2 \wedge S_2 \subseteq S_1$, and each constraint of the form $S_1 \subseteq S_2$ as $|\, S_1 \setminus S_2 \,| = 0$. Finally, use sums over the integer variables representing the Venn regions to rewrite the cardinality constraints. This reduction is simple to understand and to implement, but always requires 2^N integer variables, where N is the number of elements in $\mathsf{vars}(\phi)$. In the following, we show how to reduce this number considerably in many cases that arise in practice.

Note that what we describe below is largely orthogonal to the NP procedure in [10]. Our results should in principle apply both to the naive procedure sketched above, and to the sparse encoding procedure in [10]. Note that the NP procedure from [10], although theoretically optimal, has so far been shown to be practically useful only for satisfiable cases. For unsatisfiable cases the sparse encoding generates quantifier-free Presburger arithmetic formulas that are difficult for current SMT solver implementations.

3.2 Decomposing Conjunctions of QFBAPA Formulas

The decision procedure presented in Section 3.1 requires many integer variables because it uses a variable for the intersection of *every* combination of set variables. The intuition behind the following result is that permutations of elements within sets that are not related by a constraint in a formula do not affect the satisfiability of that formula.

Theorem 2. *Let* ϕ_1 *and* ϕ_2 *be two QFBAPA formulas. Let* V_1 *and* V_2 *denote the sets of set variables appearing in* ϕ_1 *and* ϕ_2*, respectively. Let* $V_S = V_1 \cap V_2$ *be the set of shared variables. Let* K_S *represent the set of shared integer variables. The QFBAPA formula* $\phi \equiv \phi_1 \wedge \phi_2$ *is satisfiable if and only if there exists a model* $\mathcal{M}_1$ *for* ϕ_1 *and a model* $\mathcal{M}_2$ *for* ϕ_2 *such that, for each integer variable* $k \in K_S$*,* $k^{\mathcal{M}_1} = k^{\mathcal{M}_2}$*, and such that* $\mathcal{M}_1 \sim_{V_S} \mathcal{M}_2$.

Proof. We construct a model $\mathcal{M}$ for ϕ by extending $\mathcal{M}_1$ to the variables in $V_2 \setminus V_S$. (The other direction is immediate.) We show that there exists a permutation f on $\mathbb{E}$ such that $f[v^{\mathcal{M}_2}] = v^{\mathcal{M}_1}$ for each Venn region $v \in \mathsf{venn}(V_S)$. In other words, f is a bijection that projects the interpretation in $\mathcal{M}_2$ of all intersections of the shared variables to their interpretation in $\mathcal{M}_1$. We construct f as follows: for each Venn region $v \in \mathsf{venn}(V_S)$, let f_v be a bijection from $v^{\mathcal{M}_2}$ to $v^{\mathcal{M}_1}$. Note that f_v always exists because $v^{\mathcal{M}_1}$ and $v^{\mathcal{M}_2}$ have the same cardinality, by $\sim_{V_S}$. Let $f^\star$ be $\bigcup_{v \in \mathsf{venn}(V_S)} f_v$. Observe that $f^\star$ is a bijection from $\mathcal{U}^{\mathcal{M}_2}$ to $\mathcal{U}^{\mathcal{M}_1}$, because $\mathsf{venn}(V_S)$ forms a partition of $\mathcal{U}$ in both models. To obtain the desired f, we can extend $f^\star$ to the domain and range $\mathbb{E}$ by taking its union with any bijection from $\mathbb{E} \setminus \mathcal{U}^{\mathcal{M}_2}$ to $\mathbb{E} \setminus \mathcal{U}^{\mathcal{M}_1}$. The model $\mathcal{M} = \mathcal{M}_1 \cup f[\mathcal{M}_2]$ is a model of ϕ_1 (trivially) and of ϕ_2 (by Theorem 1), therefore, it is a model of ϕ.

The following result is a generalization of Theorem 2 for a conjunction of arbitrarily many constraints.

Theorem 3. *Let $\phi_1, \ldots, \phi_n$ be n QFBAPA formulas. Let V_i denote $\mathsf{vars}(\phi_i)$ for $i \in \{1, \ldots, n\}$. Let*

$$V_S = \bigcup_{1 \leq i < j \leq n} V_i \cap V_j$$

be the set of all variables that appear in at least two sets V_i and V_j. The formula $\phi_1 \wedge \ldots \wedge \phi_n$ is satisfiable if and only if: 1) there exist models $\mathcal{M}_1$, …, $\mathcal{M}_n$ such that, for each i, $\mathcal{M}_i \models \phi_i$ and 2) there exists an interpretation $\mathcal{M}$ of the variables V_S such that $\mathcal{M} \sim_{V_S \cap V_i} \mathcal{M}_i$ for each $1 \leq i \leq n$.

(Note that the conditions imply that $|\mathcal{U}^{\mathcal{M}}| = |\mathcal{U}^{\mathcal{M}_i}|$ for each i.) The proof follows the idea of the proof of Theorem 2: it suffices to show that one can extend the model $\mathcal{M}$ to each model $\mathcal{M}_i$ by finding a bijection f_i. Note that f_i is guaranteed to exist because $\mathcal{M} \sim_{V_S \cap V_i} \mathcal{M}_i$.

Remark. It is not sufficient that for each $0 \leq i < j \leq n$, $\mathcal{M}_i \sim_{(V_i \cap V_j)} \mathcal{M}_j$, that is, that there exists bijections between all pairs of models $\mathcal{M}_i$ and $\mathcal{M}_j$. A simple counter-example is given by the (unsatisfiable) constraints:

$$\begin{aligned}
\phi_1 &\equiv |A| = 1 \wedge |B| = 1 \wedge |A \cap B| = 1 \\
\phi_2 &\equiv |A| = 1 \wedge |C| = 1 \wedge |A \cap C| = 1 \\
\phi_3 &\equiv |B| = 1 \wedge |C| = 1 \wedge |B \cap C| = 0
\end{aligned}$$

Although any conjunction $\phi_i \wedge \phi_j$ can be shown satisfiable using Theorem 2, the conjunction $\phi_1 \wedge \phi_2 \wedge \phi_3$ is unsatisfiable, because a common model $\mathcal{M}$ cannot be built from models for each three constraints.

Lemma 1. *Let $\phi_1, \ldots, \phi_n$ be n QFBAPA formulas, and let $V_1, \ldots, V_n, V_S$ be defined as above. Assume there exist $\mathcal{M}, \mathcal{M}_1, \ldots, \mathcal{M}_n$ satisfying the conditions of Theorem 3 and the additional condition that:*

$$|\mathcal{U}^{\mathcal{M}}| \geq |\bigcup_{S \in V_S} S^{\mathcal{M}}| + \sum_{i=1}^{n} |\bigcup_{S \in V_i \setminus V_S} S^{\mathcal{M}_i}|$$

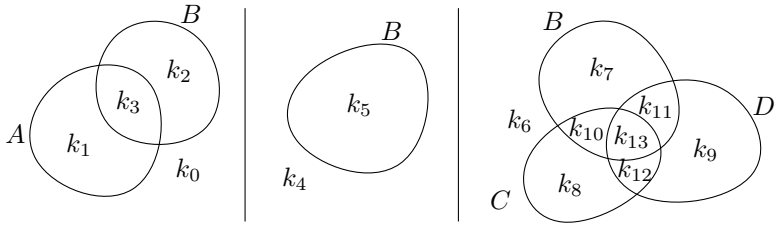

Fig. 2. Independent naming of the Venn regions of set variables $V_1 = \{A, B\}$, $V_S = \{B\}$, and $V_2 = \{B, C, D\}$.

Then there exists a model $\mathcal{M}_d$ such that, for any two sets $S_i \in V_i \setminus V_S$ and $S_j \in V_j \setminus V_S$ (with $i \neq j$), if (1) $\forall S \in V_S \cap V_i \;.\; \mathcal{M}_i \models S_i \neq S$ and (2) $\forall S \in V_S \cap V_j \;.\; \mathcal{M}_j \models S_j \neq S$, and (3) either $\mathcal{M}_i \models S_i \neq \emptyset$ or $\mathcal{M}_j \models S_j \neq \emptyset$, then $\mathcal{M}_d \models S_i \neq S_j$.

Intuitively, Lemma 1 states that for two sets variables that appear in different constraints and that are not shared, if there exists a model in which these sets are not empty and are not equal to some shared set variable, then there exists a model in which the two sets are not equal. The correctness follows from the fact that with $\mathcal{U}^{\mathcal{M}}$ sufficiently large, it is always possible to find bijections f_i such that for each i the non-shared sets are mapped to a different subset of $\mathcal{U}$. We omit the details of the proof in the interest of space.

Lemma 2. *The additional conditions of Lemma 1 are fulfilled when $\mathbb{E}$ is infinite and $|\mathcal{U}|$ is not constrained in any formula ϕ_i.*

This result is important for the combination of QFBAPA with other theories, as explained in Section 4. The proof follows from the fact that with an infinite domain $\mathbb{E}$ and an unconstrained universe $\mathcal{U}$, if there exists a model, we can extend it to a model $\mathcal{M}$ where $|\mathcal{U}^{\mathcal{M}}|$ is sufficiently large.

A decision procedure based on decompositions. Theorem 3 yields a decision procedure for the satisfiability of conjunctions of QFBAPA constraints $\phi_1 \wedge \ldots \wedge \phi_n$: independently for ϕ_1 to ϕ_n, introduce integer variables for the regions of $\mathsf{venn}(V_1)$, ..., $\mathsf{venn}(V_n)$, respectively, then introduce fresh variables for the regions of $\mathsf{venn}(V_S)$, where V_S is computed as in the theorem. Finally, constrain the sums of variables representing the same regions to be equal. (This ensures that the bijections f_i can be constructed in the satisfiable case.) As an example, consider the formula

$$|A \setminus B| > |A \cap B| \wedge B \cap C \cap D = \emptyset \wedge |B \setminus D| > |B \setminus C|$$

Let ϕ_1 be the first conjunct and ϕ_2 the other two. We have $V_1 = \{A, B\}$ and $V_2 = \{B, C, D\}$. Using the naming for regions shown in Figure 2, we obtain the

integer constraints

$$\begin{aligned} & k_1 > k_3 && \text{reduction of } \phi_1 \\ \wedge\ & k_{13} = 0 \wedge k_7 + k_{10} > k_7 + k_{11} && \text{reduction of } \phi_2 \\ \wedge\ & k_4 = k_0 + k_1 = k_6 + k_8 + k_9 + k_{12} && \text{representations of } |\, B^{\mathsf{c}} \,| \\ \wedge\ & k_5 = k_2 + k_3 = k_7 + k_{10} + k_{11} + k_{13} && \text{representations of } |\, B \,| \end{aligned}$$

A possible satisfying assignment for the integer variables is $k_4 = 2$, $k_0 = k_1 = k_2 = k_5 = k_8 = k_9 = k_{10} = 1$, and $k_3 = k_6 = k_7 = k_{11} = k_{12} = k_{13} = 0$. From these values, we can build the assignment to set variables

$$\begin{aligned} & \mathcal{U}_1 = \{e_1, e_2, e_3\}, A = \{e_1\}, B = \{e_2\} && \text{for } \phi_1 \\ & \mathcal{U}_2 = \{e_4, e_5, e_6\}, B = \{e_4\}, C = \{e_4, e_5\}, D = \{e_6\} && \text{for } \phi_2 \end{aligned}$$

Finally, to build a model for $\phi_1 \wedge \phi_2$, we need to construct two bijections f_1 and f_2 such that $f_1(e_4) = f_2(e_2)$ and $f_1[\{e_5, e_6\}] = f_2[\{e_1, e_3\}]$. This is always possible in this decision procedure, because we constrain the sizes of the Venn regions of shared variables to be equal. Intuitively, the freedom in the construction for this example comes from the fact that when we merge the models, the overlapping between C or D and A is unconstrained: we can choose to map either e_5 or e_6 to the same element as e_1, in which case either C or D will share an element with A. So one possible model for ϕ is $A = \{e_1\}, B = \{e_2\}, C = \{e_1, e_2\}, D = \{e_3\}$. (Here we chose the identity function for f_1.) Note that this model also satisfies the property of models described in Lemma 1, since, in this interpretation, $A \neq C$ and $A \neq D$.

3.3 Hypertree Decompositions

In general, and as in the previous example, the constraints ϕ_i in Theorem 3 can contain top-level conjunctions. It is thus in principle possible to decompose them further. In this section, we introduce a hypergraph representation of constraints from which it is straightforward to apply the decision procedure presented above recursively.

Definition 4 (Hypertree Decomposition). *Let ϕ be a QFBAPA formula and let $V = \mathsf{vars}(\phi)$. Let $\phi_1, \ldots, \phi_n$ be top-level conjuncts of ϕ and $V_1, \ldots, V_n$ the sets of variables appearing in them, respectively. We assume without loss of generality that $V_i \neq V_j$ for $i \neq j$. (If two constraints ϕ_i and ϕ_j have the same set of variables, consider instead the constraint $\phi_i \wedge \phi_j$.) Let $\boldsymbol{V}$ denote $\{V_1, \ldots, V_n\}$. Let $\mathcal{H} = (\boldsymbol{V}, \boldsymbol{E})$ be a hypergraph on the set of sets of variables $\boldsymbol{V}$ (so $\boldsymbol{E} \subseteq 2^{\boldsymbol{V}}$). Let $l : \boldsymbol{E} \to 2^V$ be a labeling function that associates to each hyperedge a subset of the set variables of ϕ (that subset need not be one of V_i). We call $\mathcal{H}$ a* hypertree decomposition of ϕ, *if the following properties hold:*

$(H1)$ $\forall V_i, V_j \,.\, V_i \cap V_j \neq \emptyset \implies \exists E \in \boldsymbol{E} \,.\, V_i \in E \wedge V_j \in E \wedge (V_i \cap V_j) \subseteq l(E)$

$(H2)$ $\forall E \in \boldsymbol{E}, \forall V \in \boldsymbol{V} \,.\, V \in E \iff V \cap l(E) \neq \emptyset$

$(H3)$ $\mathcal{H}$ *is acyclic*[1]

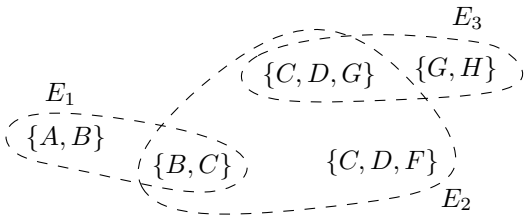

Fig. 3. A hypertree decomposition of (1). The labeling is given by: $l(E_1) = \{B\}$, $l(E_2) = \{C, D\}$, $l(E_3) = \{G\}$.

Property $(H1)$ states that two nodes that share at least one variable must be connected with an hyperedge whose label contains at least their shared variables, and Property $(H2)$ states that a node is contained in a hyperedge if and only if the label of that hyperedge shares at least one set variable with the node. Note that a formula ϕ admits in general many distinct hypertree decompositions. For instance, the hypergraph $\mathcal{H} = (\boldsymbol{V}, E)$ with a single hyperedge $E = \boldsymbol{V}$ and $l(E) = \mathsf{vars}(\phi)$ always satisfies Definition 4. To illustrate hypertree decompositions, consider the (unsatisfiable) formula

$$| A \cup B | \leq 3 \wedge C \subseteq B \wedge | (C \cap D) \setminus F | = 2 \wedge | (C \cap G) \setminus D | = 2 \wedge H \subseteq G \quad (1)$$

If we number the conjuncts from 1 to 5, we have $V_1 = \{A, B\}$, $V_2 = \{B, C\}$, $V_3 = \{C, D, F\}$, $V_4 = \{C, D, G\}$, $V_5 = \{G, H\}$. Figure 3 shows a possible hypertree decomposition of (1). The intuition behind hypertree decompositions is that hyperedges represent set of constraints, and that models for these sets need to agree only on the variables they share in their common nodes.

Reduction to integer arithmetic from hypertrees. We show now how to adapt the decision procedure presented at the end of Section 3.2 to hypertree decompositions. We proceed as follows: for each set of set variables V_i (each node in the hypertree), we introduce integer variables for all regions $\mathsf{venn}(V_i)$. Then, for each hyperedge E, we add variables for the regions $\mathsf{venn}(l(E))$. Finally, for each node V_i and each hyperedge E such that $V_i \in l(E)$, we constrain the sums of variables describing the same Venn regions to be equal. For each $v \in \mathsf{venn}(V_i \cap l(E))$, we generate the constraint:

$$\sum_{\substack{v_1 \in \mathsf{venn}(V_i) \\ \text{s.t.} v_1 \subseteq v}} k_{V_i}(v_1) = \sum_{\substack{v_2 \in \mathsf{venn}(l(E)) \\ \text{s.t.} v_2 \subseteq v}} k_{l(E)}(v_2)$$

where $k_{V_i}(\cdot)$ and $k_{l(E)}(\cdot)$ denote the integer variable representing a Venn region in the naming scheme for the Venn regions of V_i and $l(E)$, respectively.

[1] We define as a cycle any set of at least two hyperedges such that there exists a node reachable from itself by traversing all hyperedges in the set. This implies in particular that no two hyperedges intersect on more that one node. Note that this is more restrictive than the definition of a hypertree as a hypergraph that admits a tree as a host graph.

This decision procedure has the same worst-time complexity as the one presented in Section 3.1; indeed, it is possible to construct a formula ϕ with a hypertree decomposition that leads to a reduction that uses up to 2^{N+1} integer variables, where N is the number of set variables in ϕ. We have found however that the reduction works well in practice (see Section 5).

4 Integration with Z3

We implemented our new decision procedure for QFBAPA constraints based on the results of Section 3 as an extension to the state-of-the-art SMT solver Z3 [14]. In this extension, sets do not range over uninterpreted elements but over the integers. The resulting prover can thus handle constraints such as

$$(S_1 = \{1, 2, 3, 4\} \wedge S_2 \subseteq S_1 \wedge S_2 \neq \emptyset) \implies |\, S_2 \,| \in S_1$$

and such as our example from the introduction. Although we are thus in effect performing non-disjoint theory combination, we were able to implement our techniques using only the mechanism of theory plugin extensions to interact with the prover. Because such extensions can add arbitrary axioms (i.e. that are not restricted to theory elements) to the logical context of Z3 at any time, this was indeed not a major limitation. We wrote our extension in Scala and used the Java Native Interface [12] to access the C API of Z3.[2] We did not observe any significant overhead in the forwarding of function calls from and to the Java virtual machine.

Reduction to the integers. We handle constraints on sets with cardinalities by reducing them to integer linear arithmetic using the translation presented in Section 3.3. Assume for now a fixed naming of Venn regions (we describe below how we maintain a hypertree and thus a naming in the presence of incremental reasoning). For each set constraint $S_1 \subseteq S_2$ or $S_1 = S_2$, we add an axiom constraining the integer representation of the set variables in the atom. For instance, using the naming of Venn regions in Figure 2, we would add for the constraint $B \cup C \subseteq D$ the implication $B \cup C \subseteq D \implies k_7 + k_8 + k_{10} = 0$. Similarly, for each cardinality term $|\, S \,|$, we add an axiom expressing the cardinality in terms of integer variables. Using the same naming as above, we would for instance add for the term $|\, C \setminus B \,|$ the equality $|\, C \setminus B \,| = k_8 + k_{12}$. These axioms enforce that whichever boolean value the SMT solver assigns to the set predicate, it will be forced to maintain for the integer variables an assignment that is consistent with the constraints on the sizes of Venn regions.

Incremental introduction of Venn regions. Our theory extension receives messages from the core solver whenever a constraint on sets or an application of the cardinality operator is added to the logical context. We maintain at all times a hypertree decomposition $\mathcal{H}$ of the conjunction of all constraints that are on the stack. Whenever a new constraint ϕ_i is pushed to the stack, we apply the following steps:

[2] Our Scala interface to Z3 is available at `http://lara.epfl.ch/dokuwiki/jniz3`.

1. We compute $V_i = \mathsf{vars}(\phi_i)$.
2. If there is a node in $\mathcal{H}$ labeled with V_i, we use the naming of Venn regions in that node to generate the axiom expressing the reduction of ϕ_i and skip the following steps.
3. Else we introduce a new node V_i and we create fresh integer variables for each region of $\mathsf{venn}(V_i)$.
4. For each node V_j in $\mathcal{H}$ such that $V_i \cap V_j \neq \emptyset$, we add a hyperedge $E = \{V_i, V_j\}$ labeled with $l(E) = V_i \cap V_j$.
5. In the new graph, we collapse any introduced cycle as follows: let $E_1, \ldots, E_n$ be the hyperedges that participate in the cycle. We introduce a new hyperedge E labeled with $\bigcup_{i \in \{1,\ldots,n\}} l(E_i)$ and whose content is the union of the contents of $E_1, \ldots, E_n$. We then remove from the graph the edges E_1 to E_n.
6. For each newly created edge E in the obtained, acyclic, hypergraph we introduce fresh integer variables to denote the regions of $\mathsf{venn}(l(E))$.
7. Finally, for each node in the new edge, we constrain the sums of integer variables that denote the same Venn regions to be equal.

Note that all introduced integer variables are additionally constrained to be non-negative. When constraints are popped from the stack:

1. We remove from $\mathcal{H}$ all nodes that were created for these constraints, if any.
2. For each hyperedge E that contained at least one removed variable, we check whether it now contains only one node, in which case we delete it.

Removing such nodes and edges provides two benefits: constraints added in a different search branch may generate a smaller (less connected) hypertree, and Z3 has the opportunity to remove from its clause database reduction axioms that have become useless. We efficiently detect cycles by maintaining equivalence classes between nodes that correspond to the connected components in the hypertree. We can then check, when we introduce a new hyperedge, whether it connects nodes from different equivalence classes, which corresponds to our definition of a cycle. We use a union-find data-structure to maintain the equivalence classes.

Interpreted elements. When considering sets in combination with other theories, it is natural to allow the use of the predicate $e \in S$ denoting that a given set contains a particular (interpreted) element, or constant sets $\{e_1, e_2\}$. For the simplicity of the argument, we assume here that all sets have the same underlying element sort $\mathbb{E}$. We handle interpreted elements such as e, e_1 and e_2 by adding to the logic two connecting functions, $\mathsf{singleton} : \mathbb{E} \to \mathsf{Set}$ and $\mathsf{element} : \mathsf{Set} \to \mathbb{E}$. We create for each interpreted element that appears in a predicate $e \in S$ or in a constant set a singleton set term $\mathsf{singleton}(e)$, and we add the axioms $|\mathsf{singleton}(e)| = 1$ and $\mathsf{element}(\mathsf{singleton}(e)) = e$. So $\mathsf{singleton}$ is interpreted as a constructor that builds a singleton set from an element, and $\mathsf{element}$ as a function that for singleton sets computes its element, and that is uninterpreted otherwise. We can thus rewrite $e \in S$ as $\mathsf{singleton}(e) \subseteq S$ and constant sets as unions of singletons. The connecting functions are used to propagate equalities: when two elements are found to be equal in their theory, congruence

closure will conclude that their singleton sets (if they exist) need to be equal as well. The other direction is covered in the following paragraph.

Communicating equalities. The reduction to integer linear arithmetic ensures that if two sets must be equal, then the sizes of the Venn regions of their symmetric difference will be constrained to be 0. However, for completeness we also need to detect that if their symmetric difference is forced to be empty by some integer constraints, then the sets must be equal. To enforce this, we could, for each pair of set variables (S_1, S_2) generate an axiom equivalent to

$$(|\, S_1 \setminus S_2 \,| = \emptyset \wedge |\, S_2 \setminus S_1 \,| = \emptyset) \implies S_1 = S_2$$

where the cardinality terms would be rewritten using the naming of set regions for S_1 and S_2. However, because not every pair of set variable appears together in some constraint, our hypertree decomposition does not in general define names for the intersections of any variables S_1 and S_2. Our solution is to generate such axioms for all pairs of variables that appear together in a node of the hypergraph. Additionally, for each set variable we generate the axiom $|\, S \,| = 0 \iff S = \emptyset$. We argue that, for sets that range over an infinite domain (such as the integers), this is sufficient: Consider two set variables S_1 and S_2 that do not appear in a common node. From lemmas 1 and 2, we have that as long as S_1 or S_2 is non-empty or distinct from a shared set variable, there exists a model in which $S_1 \neq S_2$. Both cases are covered by the introduced axioms. Indeed, even if the two variables are equal to a different shared variable, by transitivity of the equality, their equality will be propagated. We thus have the property that any two set variables that are not constrained to be equal can be set to be distinct in a model. This is consistent with the theory combination approach taken in Z3 [13]. When we construct the hypergraph as presented above, we also make sure that all terms representing singleton sets (that is, applications of the $\mathsf{singleton}(\cdot)$ function) share a common hyperedge, even if they don't syntactically appear in a common constraint. This ensures that all equalities between finite sets induced by cardinality constraints will result in equalities between their elements through Z3's congruence closure algorithm, and will be communicated to the proper theory solver.

5 Evaluation

We evaluated our implementation using benchmarks from two verification sources. Figure 4 shows our experimental results.[3]

Jahob benchmarks. We included all benchmarks from [10] in our evaluation. These formulas express verification conditions generated with the Jahob system [7] for programs manipulating (abstractions of) pointer-based data-structures such as linked lists. In [10], the benchmarks were used to compare the efficiency of the sparse encoding into linear arithmetic with the explicit one

[3] All benchmarks and the set of axioms we used are available from `http://lara.epfl.ch/~psuter/vmcai2011/`.

Benchmark	status	sets	cons.	Venn regs.	prop.	prev. best	our time
CADE07-1	unsat	1	1	2	1.00	< 0.1	< 0.1
CADE07-2	unsat	2	1	4	1.00	< 0.1	< 0.1
CADE07-2a	unsat	6	5	28	0.44	1.8	< 0.1
CADE07-2b	sat	6	5	28	0.44	< 0.1	< 0.1
CADE07-3	unsat	2	1	4	1.00	< 0.1	< 0.1
CADE07-3a	unsat	5	4	16	0.50	0.4	< 0.1
CADE07-3b	sat	5	4	16	0.50	< 0.1	< 0.1
CADE07-4	unsat	5	5	80	2.50	0.5	< 0.1
CADE07-4b	sat	5	5	76	2.38	0.1	< 0.1
CADE07-5	unsat	7	5	168	1.31	13.6	< 0.1
CADE07-5b	sat	7	5	168	1.31	0.4	< 0.1
CADE07-6	unsat	6	6	32	0.50	0.4	< 0.1
CADE07-6a	unsat	16	20	596	0.01	> 100	0.3
CADE07-6b	sat	16	22	1120	0.02	0.8	0.3
CADE07-6c	sat	16	20	596	0.01	0.9	0.4
listContent	sat	3	4	14	1.75		< 0.1
listContent-ax	unsat	4	5	16	1.00		< 0.1
listReverse	sat	6	6	26	0.41		< 0.1
listReverse-ax	unsat	9	11	308	0.60		0.2
setToList	sat	6	8	76	1.19		< 0.1
setToList-ax	unsat	7	9	114	0.89		< 0.1
listConcat	sat	11	19	54	0.03		0.2
listConcat-ax	unsat	28	33	268	< 0.01		0.4
treeContent	sat	4	5	24	1.50		0.1
treeContent-ax	unsat	6	7	28	0.44		0.1
treeMirror	sat	6	6	20	0.31		< 0.1
treeMirror-ax	unsat	11	11	572	0.28		0.3
treeToList	sat	8	10	24	0.09		0.1
treeToList-ax	unsat	29	35	2362	< 0.01		1.7

Fig. 4. Experimental results. The column "sets" is the number of set variables, "cons." is the maximal number of constraints on the stack, "Venn regs." is the maximal number of distinct Venn regions created from the hypertree structure, "prop." is the proportion of created Venn regions compared to the 2^N naive naming scheme and is a measure of the efficiency of our decomposition technique, "prev. best" indicates the previously best solving time, and "time" is the running time with the new implementation. All times are in seconds. The experiment was conducted using a 2.66GHz Quad-core machine, and using Z3 2.11.

(as in Section 3.1). We indicate for these benchmarks the previously best time using either method. It is important to note that it was shown in [10] that the two methods were complementary: the sparse encoding outperformed the explicit one for sat instances and vice-versa. We do not claim that the absolute difference in time is a significant measure of our algorithmic improvements, but rather provide these numbers to illustrate that our new techniques can be used to efficiently handle sat and unsat instances.

Functional programs. We also included new benchmarks consisting of verification conditions for Scala functions without side effects. Additionally to sets and elements, these examples contain constraints on algebraic data types and functions symbols. Each verification condition contains at least one (recursive) function call. We first treated these function symbols as uninterpreted. We then used universally quantified axioms to define the interpretation of the functions and used Z3's pattern-based instantiation mechanism to apply the axioms. Without the axioms, all formulas were invalid (sat), and with the axioms, they were all proved valid.

To the best of our knowledge, the only previous implementation that is complete for QFBAPA and reports performance on benchmarks is [10]. We show significant improvement over the existing benchmarks. While results in [10] were unable to handle unsatisfiable formulas with 16 variables, we report success on formulas with 29 variables, which were automatically generated from verification of functional programs.

6 Related Work

The complexity of QFBAPA was settled in [10] to be NP. The complexity of the quantified case as well as the first quantifier elimination implementation was described in [8]. The decidability of the logic itself dates back at least to [3].

The work of combination of QFBAPA with other decidable theories has so far included implementation as part of the Jahob system [7], which requires manual decomposition steps to be complete [20, 21]. A complete methodology for using QFBAPA as a glue logic for non-disjoint combination was introduced in [17], with additional useful cases introduced in [16, 18], some of which are surveyed in [9]. The combination method we describe is simple, in that it does not require exchanging set constraints between different theories that share sets of objects. In that sense, it corresponds to the multi-sorted combination setup of [19], which introduces a non-deterministic procedure that was, to the best of our knowledge, not implemented. Combinations of theories that have finite domains has been explored in [6]. In this paper we have focused on combinations with integers, but we believe that our approach can be adapted to more general cases.

Our decomposition of formulas was inspired by the algorithms for bounded (hyper)tree width from constraint satisfaction [4], although we do not directly follow any particular decomposition algorithm from the literature. These algorithms are typically used to reduce subclasses of NP-hard constraint satisfaction problems over finite domains to polynomial-time algorithms. To the best of our knowledge, they have not been applied before to satisfiability of sets with cardinality operators. Our results suggest that this approach is very promising and we expect it to extend to richer logics containing QFBAPA, such as [18].

Research in program analysis has used cardinality constraints in abstract domains [5, 15], typically avoiding the need for a full-fledged QFBAPA decision procedure. Thanks to our efficient implementation of QFBAPA, we expect that

precise predicate abstraction approaches [1] will now also be able to use QFBAPA constraints.

Acknowledgments. We thank Nikolaj Bjørner and Leonardo de Moura for their help with Z3.

References

1. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker Blast. STTT 9(5-6), 505–525 (2007)
2. Dewar, R.K.: Programming by refinement, as exemplified by the SETL representation sublanguage. ACM TOPLAS (July 1979)
3. Feferman, S., Vaught, R.L.: The first order properties of products of algebraic systems. Fundamenta Mathematicae 47, 57–103 (1959)
4. Gottlob, G., Greco, G., Marnette, B.: Hyperconsistency width for constraint satisfaction: Algorithms and complexity results. In: Graph Theory, Computational Intelligence and Thought. LNCS, vol. 5420, pp. 87–89 (2009)
5. Gulwani, S., Lev-Ami, T., Sagiv, M.: A combination framework for tracking partition sizes. In: POPL. pp. 239–251 (2009)
6. Krstic, S., Goel, A., Grundy, J., Tinelli, C.: Combined satisfiability modulo parametric theories. In: TACAS. LNCS, vol. 4424, pp. 602–617 (2007)
7. Kuncak, V.: Modular Data Structure Verification. Ph.D. thesis, EECS Department, Massachusetts Institute of Technology (February 2007)
8. Kuncak, V., Nguyen, H.H., Rinard, M.: Deciding Boolean Algebra with Presburger Arithmetic. J. of Automated Reasoning (2006)
9. Kuncak, V., Piskac, R., Suter, P., Wies, T.: Building a calculus of data structures. In: VMCAI. LNCS, vol. 5944, pp. 26–44 (2010)
10. Kuncak, V., Rinard, M.: Towards efficient satisfiability checking for Boolean Algebra with Presburger Arithmetic. In: CADE-21. LNCS, vol. 4603 (2007)
11. Lam, P., Kuncak, V., Rinard, M.: Generalized typestate checking for data structure consistency. In: VMCAI. LNCS, vol. 3385, pp. 430–447 (2005)
12. Liang, S.: The Java Native Interface: Programmer's Guide and Specification. Addison-Wesley (1999)
13. de Moura, L., Bjørner, N.: Model-based theory combination. Electronic Notes in Theoretical Computer Science 198(2), 37–49 (2008)
14. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS. LNCS, vol. 4963, pp. 337–340 (2008)
15. Pérez, J.A.N., Rybalchenko, A., Singh, A.: Cardinality abstraction for declarative networking applications. In: CAV. LNCS, vol. 5643, pp. 584–598 (2009)
16. Suter, P., Dotta, M., Kuncak, V.: Decision procedures for algebraic data types with abstractions. In: POPL (2010)
17. Wies, T., Piskac, R., Kuncak, V.: Combining theories with shared set operations. In: FroCoS: Frontiers in Combining Systems. LNCS, vol. 5749, pp. 366–382 (2009)
18. Yessenov, K., Kuncak, V., Piskac, R.: Collections, cardinalities, and relations. In: VMCAI. LNCS, vol. 5944, pp. 380–395 (2010)
19. Zarba, C.G.: Combining sets with integers. In: FroCoS: Frontiers of Combining Systems. LNCS, vol. 2309, pp. 103–116 (2002)
20. Zee, K., Kuncak, V., Rinard, M.: Full functional verification of linked data structures. In: PLDI. pp. 349–361 (2008)
21. Zee, K., Kuncak, V., Rinard, M.: An integrated proof language for imperative programs. In: PLDI. pp. 338–351 (2009)