On Set-Driven Combination of Logics and Verifiers

LARA Technical Report, February 2, 2009

Viktor Kuncak[1] and Thomas Wies[1,2]

[1] EPFL School of Computer and Communication Sciences, Switzerland
[2] Department of Computer Science at University of Freiburg, Germany

Abstract. We explore the problem of automated reasoning about the non-disjoint combination of logics that share set variables and operations. We prove a general combination theorem, and apply it to show the decidability for the quantifier-free combination of formulas in WS2S, two-varible logic with counting, and Boolean Algebra with Presburger Arithmetic.
Furthermore, we present an over-approximating algorithm that uses such combined logics to synthesize universally quantified invariants of infinite-state systems. The algorithm simultaneously synthesizes loop invariants of interest, and infers the relationships between sets to exchange the information between logics. We have implemented this algorithm and used it to prove detailed correctness properties of operations of linked data structure implementations.

1 Introduction

Automated abstraction techniques such as predicate abstraction are among the most promising approaches for verifying systems with large state spaces [1–3,6,9,10,16,17]. Such techniques were enabled by the recent progress in SAT and SMT solvers [4, 12, 14, 26]. The range of reachability invariants that can be inferred by such approaches depends on the expressive power of the logics supported by the SMT solvers. Current SMT solvers implement the disjoint combination of quantifier-free theories, in essence following the approach pioneered by Nelson and Oppen [30]. Such solvers serve as decision procedures for quantifier-free formulas, typically containing uninterpreted function symbols, linear arithmetic, and bitvectors. The limited expressiveness of SMT prover logics translates into a limited class of invariants that automated abstraction-based tools can infer.

To enable broader applications of automated abstraction techniques, this paper considers decision procedures for combination of quantified formulas in non-disjoint theories. The idea of combining rich theories within an expressive language has been explored in interactive provers [5, 7, 29, 31]. Such integration efforts are very useful, but do not result in complete decision procedures for the combined logics. The study of completeness for non-disjoint combination is relatively recent [37, 40] and provided foundation for the general problem. Our paper considers a particular combination of non-disjoint theories–theories sharing operations on *sets of uninterpreted elements*. To the best of our knowledge, this problem has not been considered before, despite the usefulness of sets for reasoning about dynamically created objects and concurrent processes.

Challenges in communicating constraints on sets. The idea of combining decision procedure is to check the satisfiability of a conjunction of formulas $A \wedge B$ by using one decision procedure, D_A, for A and another decision procedure, D_B, for B. To obtain a complete decision procedure, D_A and D_B must communicate to ensure that a model found by D_A and a model found by D_B can be merged into a model for $A \wedge B$. Craig's interpolation theorem for first-order logic implies that if $A \wedge B$ is unsatisfiable, then there exists an interpolant I such that

1. $A \rightarrow I$ is valid
2. $I \wedge B$ is unsatisfiable, and
3. I is a (potentially quantified) first-order formula containing only predicate symbols and variables common to A and B.

The interpolant I can be used to communicate the information between D_A and D_B. When A and B belong to disjoint theories, I contains equalities as the only kind of atomic formulas. The class of such formulas admits quantifier elimination, so there are only finitely many non-equivalent formulas I, making it easier to construct a complete combined decision procedure.

The combination problem is more difficult for formulas that share sets of elements, because there are infinitely many constraints on sets definable in typical logics. For example, for every non-negative integer K, most logics can express the property that a shared set has exactly K elements. The set of possible interpolants I is thus not bounded by the number of symbols shared between A and B, but depends also on the structure of formulas A and B.

Decision procedure based on projections. In this situation we suggest that D_A computes the projection S_A of A onto shared set variables. This projection is equivalent to existentially quantifying over predicates and variables appearing in A but not in B, and corresponds to the strongest interpolant. D_B can similarly compute the projection S_B of B. This reduces the problem to checking the satisfiability of $A \wedge B$ to satisfiability of a formula with sets $S_A \wedge S_B$. (Alternatively, D_B could attempt to directly check $S_A \wedge B$.)

A logic for shared constraints on sets. The logic of sets used to express the projections S_A and S_B is a key parameter of such a combination approach, and depends on the logics of formulas A, B. Inspired by verification of linked data structures, we consider as the logics for A, B weak monadic second-order logic of two successors WS2S [36], two-variable logic with counting C^2 [34], and BAPA [23]. Remarkably, in each of these cases, *the smallest logic needed to express the projection formulas has the expressive power of Quantifier-Free Boolean Algebra with Presburger Arithmetic (QFBAPA)* [24], Figure 1. We also show that the decision procedures for these logics can be naturally extended to compute QFBAPA projections, with complexity of projection no higher than the complexity of the satisfiability problem. Given that QFBAPA has been shown NP-complete [24], we believe that it is an ideal candidate as a common reduction target for expressive logics sharing sets.

Decidability results. Using the approach above, we obtain a decidable logic that combines WS2S, C^2, and BAPA. We have found this logic to be useful for proving verification conditions arising in data structure verification.

$$
\begin{array}{l}
F ::= A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg F \\
A ::= B_1 = B_2 \mid B_1 \subseteq B_2 \mid T_1 = T_2 \mid T_1 < T_2 \mid K \,\mathsf{dvd}\, T \\
B ::= x \mid \emptyset \mid \mathcal{U} \mid B_1 \cup B_2 \mid B_1 \cap B_2 \mid B^c \\
T ::= k \mid K \mid \mathsf{CARDUNIV} \mid T_1 + T_2 \mid K \cdot T \mid \ |B| \\
K ::= \ldots -2 \mid -1 \mid 0 \mid 1 \mid 2 \ldots
\end{array}
$$

Fig. 1. Quantifier-Free Boolean Algebra with Presburger Arithmetic (QFBAPA)

Uses in verification. We consider practical consequences of this result for verification of infinite-state systems. Challenges in deploying such decision procedure include 1) reusing existing automated reasoning tools, and 2) inferring loop invariants. We present a verification approach that uses the same mechanism to address both challenges. The approach infers loop invariants using Boolean heaps [33]. Moreover, these loop invariants are constraints on certain sets of objects, and they provide information about the projections on set variables. Even though we cannot expect the approach to be complete, we have used it to verify properties that involve transitive closure, non-tree fields, and cardinality in lists and trees, without supplying loop invariants or lemmas. We are not aware of any other system that can verify such examples with such degree of automation. The closest approach that we are aware of is [41]. Compared to [41], the approach we present does not necessarily require explicitly naming sets of objects, supplying lemmas about changes to such sets, or specifying loop invariants.

Contributions. This paper makes the following contributions:

1. We present a simple technique for showing decidability of theories that share sets of elements, and show that the logics
   (a) weak monadic second-order logic of two successors WS2S [36];
   (b) two-variable logic with counting C^2 [34];
   (c) Boolean Algebra with Presburger Arithmetic [13, 23, 24]
   meet the conditions of the technique, which allows the use of their combination in verification.
2. We present an approach for verifying program properties expressed in this logic. The approach combines synthesis of shared formulas with loop invariant inference. We have implemented this approach and found that it is useful in practice.

2 Example

2.1 Verifying a Code Fragment

Our example illustrates a formula arising from verifying unbounded linked data structures, and explains why our combination technique is complete for proving the validity of an interesting class of such formulas. Figure 2 shows our example, which is a fragment of Java code for insertion into a binary search tree, factored out into a separate `insertAt` method (we also verified the full code, containing loops). The search tree has fields (`left`, `right`) that form a tree, and field `data`, which is not necessarily an injective function (an element may be stored multiple times in the tree). The `insertAt` method is meant to be invoked when the insertion procedure has found a node `p` that

```
class Node {Node left,right; Object data;}
class Tree {
    private static Node root;
    private static int size; /*:
    private static specvar nodes :: objset;
    vardefs "nodes=={x. (root,x) ∈ {(x,y). left x = y ∨ right x = y}*}";
    private static specvar content :: objset;
    vardefs "content=={x. ∃ n. n ≠ null ∧ n ∈ nodes ∧ data n = x} " */

    private void insertAt(Node p, Object e) /*:
      requires "tree [ left , right ] ∧ nodes ⊆ Object.alloc ∧ size = card content ∧
                e ∉ content ∧ e ≠ null ∧ p ∈ nodes ∧ p ≠ null ∧ left p = null"
      modifies nodes,content,left, right ,data,size
      ensures "size = card content" */
    {
        Node tmp = new Node();
        tmp.data = e;
        p. left = tmp;
        size = size + 1;
    }
}
```

Fig. 2. Fragment of insertion into a tree

```
tree[ left , right ] ∧ left p = null ∧ p ∈ nodes ∧
nodes={x. (root,x) ∈ {(x,y). left x = y| right x = y}^*} ∧
content={x. ∃ n. n ≠ null ∧ n ∈ nodes ∧ data n = x} ∧
e ∉ content ∧ nodes ⊆ alloc ∧
tmp ∉ alloc ∧ left tmp = null ∧ right tmp = null ∧
data tmp = null ∧ (∀ y. data y ≠ tmp) ∧
nodes1={x. (root,x) ∈ {(x,y). ( left (p:=tmp)) x = y) | right x = y} ∧
content1={x. ∃ n. n ≠ null ∧ n ∈ nodes1 ∧ (data(tmp:=e)) n = x} ⟶
          card content1 = card content + 1
```

Fig. 3. Verification condition for Fig. 2

has no left child. It inserts the given object `e` into a fresh node `tmp` that becomes the new left child of `p`.

Specification and verification in Jahob. In addition to Java statements, the example in Fig. 2 contains preconditions and postconditions, written in the notation of the Jahob verification system [21, 38, 41]. The *vardefs* notation introduces two sets: 1) the set of auxiliary objects *nodes*, denoting the `Node` objects stored in the binary tree, and 2) the set *content* denoting the useful content of the tree. To verify such examples in the previously reported approach [41], the user of the system had to manually provide the definitions of such sets, and to manually introduce certain lemmas describing changes to these sets. Our decidability result means that there is no need to manually introduce such lemmas.

Decidability of the verification condition. Figure 3 shows the verification condition formula for method `insertAt`. The validity of this formula implies that invoking a method in a state satisfying the precondition results in a state that satisfies the postcon-

SHARED SETS: nodes, nodes1, content, content1, {e}, {tmp}

WS2S FRAGMENT:
tree[left , right] $\wedge$ left p = null $\wedge$ p $\in$ nodes $\wedge$ left tmp = null $\wedge$ right tmp = null $\wedge$
nodes={x. (root,x) $\in$ {(x,y). left x = y | right x = y}^* } $\wedge$
nodes1={x. (root,x) $\in$ {(x,y). (left (p:=tmp)) x = y) | right x = y}
CONSEQUENCE: nodes1=nodes $\cup$ {tmp}

C2 FRAGMENT:
data tmp = null $\wedge$ ($\forall$ y. data y $\neq$ tmp) $\wedge$ tmp $\notin$ alloc $\wedge$ nodes $\subseteq$ alloc $\wedge$
content={x. $\exists$ n. n $\neq$ null $\wedge$ n $\in$ nodes $\wedge$ data n = x} $\wedge$
content1={x. $\exists$ n. n $\neq$ null $\wedge$ n $\in$ nodes1 $\wedge$ (data(tmp:=e)) n = x}
CONSEQUENCE: nodes1 $\neq$ nodes $\cup$ {tmp} $\vee$ content1 = content $\cup$ {e}

BAPA FRAGMENT: e $\notin$ content $\wedge$ card content1 $\neq$ card content + 1
CONSEQUENCE: e $\notin$ content $\wedge$ card content1 $\neq$ card content + 1

Fig. 4. Separated conjuncts for negation of Fig. 3, with consequences about shared sets

dition of `insertAt`. The formula contains the transitive closure operator, quantifiers, set comprehensions, and the cardinality operator. Nevertheless, there is a (syntactically defined) decidable class of formulas that contains the verification condition in Fig. 3. This decidable class is a set-sharing combination of three decidable logics, and can be decided using the method we present in this paper.

To understand the method for proving the formula in Fig. 3, consider the problem of showing the unsatisfiability of the negation of the formula. Figure 4 shows the conjuncts of the negation, grouped according to three decidable logics to which the conjuncts belong: 1) weak monadic second-order logic of two successors WS2S [36], 2) two-variable logic with counting C^2 [34], and 3) Boolean Algebra with Presburger Arithmetic (BAPA) [13, 23, 24]. For the formula in each of the fragments, Fig. 4 also shows a consequence formula that contains only shared sets and statements about their cardinalities. (We represent elements as singleton sets, so we admit formulas sharing elements as well. Cardinality constraints appear already with sets of elements and we believe our approach could be combined with [20].)

A decision procedure. Note that the conjunction of the consequences of three formula fragments is an unsatisfiable formula. This shows that the original verification condition is valid. In general, our decidability result shows that the decision procedures of logics such as WS2S and C^2 can be naturally extended to compute strongest consequences of formulas involving given shared sets. These consequences are expressed in quantifier-free BAPA , which is NP-complete [24]. One possible decision procedure for satisfiability of combined formulas is

1. split the formula into fragments (belonging to WS2S, C^2, or BAPA);
2. for each fragment compute its strongest QFBAPA consequence;
3. check the satisfiability of the conjunction of consequences.

Remarks on embedding into HOL. We use higher-order logic (HOL) notation for the verification conditions, so each of the decidable fragments is an embedding of the decidable logic into HOL. To see the correspondence of these fragments with the standard notation of these decidable logics, note that set comprehensions can be eliminated

```
public static void add(Object e) {
  Node n = root, p = null;
  boolean wentLeft = false;
  while (n != null) {
    p = n;
    //: havoc wentLeft
    if (wentLeft) n = n.left;
    else n = n.right;
  }
  Node tmp = new Node();
  tmp.data = e;
  if (p == null) root = tmp;
  else if (wentLeft) p.left = tmp;
  else p.right = tmp;
  size = size + 1;
}
```

LOOP CUTPOINT:
(p = null ∧ n = null ⟶ root = null) ∧
(p ≠ null ∧ wentLeft ⟶ left p = n) ∧
(p ≠ null ∧ ¬wentLeft ⟶ right p = n) ∧
p ∈ content ∧
n ∈ content ∧
content = old content ∧
nodes = old nodes

RETURN POINT:
nodes = old nodes ∪ {tmp} ∧
content = old content ∪ {e}

Fig. 5. Tree insertion and invariants inferred for the loop and the return point

using universal quantifiers, and that the $:=$ operator (denoting the standard function update) can be eliminated using case analysis. For the WS2S fragment we assume an embedding such as [39], which requires a conjunct such as $tree[left, right]$ that interprets function symbols as appropriate successors, and encodes the transitive closure using monadic second-order quantification. We assume that the constant $null$ has the property $f\,null = null$ for each of the fields $left, right, data$. For C^2, the embedding into HOL uses the function $data$ to denote a binary predicate with the property $\forall x.\exists^{=1} y.\, data(x, y)$, and uses unary predicates to represent sets. For example, the set definition

content1={x. ∃ n. n ≠ null ∧ n ∈ nodes1 ∧ (data(tmp:=e)) n = x}

from Fig. 4 corresponds to the two-variable logic formula

$$\forall x.\mathsf{content1}(x) \leftrightarrow (\exists n.n \neq \mathsf{null} \wedge \mathsf{nodes1}(n) \wedge ((n = \mathsf{tmp} \wedge x = e) \vee (n \neq \mathsf{tmp} \wedge \mathsf{data}(n, e))))$$

In the above formula, the only variables are x and n; the symbols $null, tmp, e$ are treated as constants.

2.2 Inferring Invariants and Shared Formulas

Figure 5 shows the tree insertion operation (abstracting the 'key' field used for sorting, which is irrelevant for this example). Our tool successfully verifies that this example preserves the representation invariants given in Fig. 2. For this purpose, the tool synthesizes invariants for the loop cutpoint and the return point of method `insert`. The important parts of the inferred invariants are shown on the right-hand side of Fig. 5. Our loop invariant inference uses symbolic shape analysis based on Boolean heaps [33, 38]. This shape analysis is an abstract interpretation whose abstract domain consists of formulas that are disjunctions of universally quantified Boolean combinations of facts that

express membership in sets such as `nodes` and `content`. The individual disjuncts (we call them abstract states in the following) are quantifier-free set algebra identities. The sets in the Boolean algebra expressions are defined in higher-order logic fragments. The analysis abstracts the concrete program on-the-fly during fixed point computation generating entailments that are checked by invoking the decision procedures. The abstraction is computed on-the-fly because it uses already inferred abstract states to communicate information between the different decision procedures. Hereby, the analysis exploits the dependencies between the definitions of the different set variables. For instance, in order to compute the invariant for the return point of method `insert`, the tool first computes the precise update on set `nodes` using the MONA decision procedure. The resulting constraint is used to compute the precise update on the set `content` (using Z3 instead of a decision procedure for C^2, which works well for these examples) and in turn this constraint is used to prove preservation of the invariant `size = card content` (using the BAPA decision procedure).

3 Deciding Formulas Sharing Set Expressions

We next explain the idea of our complete decision procedure for formulas in decidable logics that share set variables. We use the framework of higher-order logic (HOL) in our presentation (The problem could alternatively be described in second-order logic or in first-order logic, in which case the first-order theories would share the operations of Boolean algebra of sets.) We assume that the universe $\mathcal{I}$ of individual elements is countably infinite. Let Fin denote the family of all finite subsets of $\mathcal{I}$. A special universal-set variable $\mathcal{U}$ ranges over elements of Fin. We assume that all set variables range only over subsets of $\mathcal{U}$. We write $\mathsf{FV}(F)$ for the set of free variables of a higher-order logic formula F.

Problem statement. We consider the truth value of the statement

$$\exists \mathcal{U} \in \mathsf{Fin}. \exists s_1, \ldots, s_p \subseteq \mathcal{U}. \exists x_1, \ldots, x_q.\ B(F_1, \ldots, F_n) \tag{1}$$

where

- $F_1, \ldots, F_n$ are HOL formulas with $\mathsf{FV}(F_i) \subseteq \{s_1, \ldots, s_p, x_1, \ldots, x_q\}$, and with each formula F_i belonging to a potentially different HOL formula set L_i. Each formula set L_i is closed under negation.
- $B(F_1, \ldots, F_n)$ denotes a formula built from $F_1, \ldots, F_n$ using only propositional operations $\wedge, \vee$.
- $s_1, \ldots, s_p \in V_S$ are set variables, whereas $x_1, \ldots, x_q \notin V_S$ are the remaining variables.
- The only variables shared between formulas F_i are set variables, that is, $\mathsf{FV}(F_i) \cap \mathsf{FV}(F_j) \subseteq V_S$ for $i \neq j$.

Note that the formula sets L_i (i.e. fragments of HOL) may contain quantified formulas. There is no loss of generality in assuming that formulas contain no variables of element type because we can represent an element a by the set $\{a\}$. Using only monotonic

operations $\wedge, \vee$ in B also does not lose generality because L_i are closed under negation. Therefore, the problem we are solving is deciding quantifier-free combinations of formulas sharing sets of elements.

For each i let $\boldsymbol{y}_i$ denote the variables from $\mathsf{FV}(F_i) \setminus V_S$ and let from now on F_i' denote $\exists \boldsymbol{y}_i.F_i$. Then $\mathsf{FV}(F_i') \subseteq V_S$. By rules for moving quantifiers, (1) is equivalent to

$$\exists \mathcal{U} \in \mathsf{Fin}. \exists s_1, \ldots, s_p \subseteq \mathcal{U}. B(F_1', \ldots, F_n') \tag{2}$$

In [22–24] we introduced the name Boolean Algebra for Presburger Arithmetic (BAPA) for a logic of formulas with set and integer variables, cardinality operator, and Presburger Arithmetic on integers. BAPA is strictly more expressive than Boolean algebra of sets, because it can express that two arbitrary set variables have the same cardinality. BAPA admits effective quantifier elimination [23]. Figure 1 shows the syntax of the quantifier-free fragment of BAPA, denoted QFBAPA.

Definition 1 (Effectively Cardinality-Linear). *We call a set L of HOL formulas* effectively cardinality-linear *if there exists an algorithm that, given a formula F in L and variables $s_1, \ldots, s_p$ denoting finite sets, constructs a QFBAPA formula equivalent to the projection $\exists x_1, \ldots, x_q.F$, where $\{x_1, \ldots, x_q\} = \mathsf{FV}(F) \setminus \{s_1, \ldots, s_p\}$.*

By a Venn region over sets $s_1, \ldots, s_p$ we mean a set $s_1^{\alpha_1} \cap \ldots \cap s_p^{\alpha_p}$ where $s_i^{\alpha_i}$ denotes either s_i or $\mathcal{U} \setminus s_i$. From properties of BAPA it follows that a formula with $\mathsf{FV}(F) = \{s_1, \ldots, s_p\}$ is equivalent to a QFBAPA formula iff there is a constraint on the cardinalities of Venn regions over $s_1, \ldots, s_n$ that is expressible in Presburger Arithmetic.

If all logics L_i are cardinality-linear fragments of HOL, then the algorithm for converting to QFBAPA along with an NP decision procedure [24] for QFBAPA gives a decision procedure for the combined logic.

Theorem 1. *If the logics L_i are effectively cardinality-linear, then the problem (1) is decidable.*

To make this result useful, we next show that several logics that we found useful in verification are effectively cardinality linear: BAPA itself, WS2S, and C^2.

BAPA is Cardinality Linear. Formulas in BAPA include the QFBAPA formulas in Figure 1, as well as formulas obtained by applying any number of universal and existential set and integer quantifiers to QFBAPA formulas. The quantifier elimination algorithm from [23] eliminates set and integer quantifiers from BAPA formulas. This algorithm therefore shows that BAPA is effectively cardinality linear.

WS2S is Cardinality-Linear. We next show effective cardinality linearity for WS2S, Weak Monadic Second-Order Logic of Two Successors. Variables in WS2S formulas range over finite subsets of nodes in an infinite binary tree. WS2S supports set algebra operations, quantification over sets and elements, as well as left and right successor functions in the tree (for definition of WS2S see e.g. [18, 19]). The observations we make in this section can be derived from [18], what is new is their use as part of our combination method.

Consider a WS2S formula F with $\mathsf{FV}(F) = \{s_1, \ldots, s_n\}$. By automata-logic connection used in the decision procedure for WS2S [19, 36], let $A(F)$ be a nondeterministic top-down tree automaton encoding the models of F and let $L(F)$ be the

regular tree language accepted by $A(F)$. A node n in a regular tree in $L(F)$ has a label that specifies for each set s_i whether $n \in s_i$ holds. Consequently, the label of a tree node specifies the Venn region to which the node belongs. The number of elements of a given Venn region is given by the number of nodes in the tree labelled by a given set of labels, which is described by the Parikh image of $L(F)$. The Parikh image of a tree language is the Parikh image of the corresponding context-free grammar, and is therefore a semilinear set [18,32], which means that it is expressible in Presburger Arithmetic [15]. To obtain a QFBAPA formula corresponding to F, it suffices to construct the tree automaton and compute its Parikh image represented as a Presburger Arithmetic formula.

We have presented WS2S as an example of an expressive logic that is cardinality linear. By the complexity of the decision procedure (and NP-completeness of QFBAPA [24]), computing the quantifier-free BAPA formula from WS2S formulas is non-elementary. For less concise logics, we obtain better complexity of the projection. One of these examples are regular expressions, whose Presburger Arithmetic formulas describing Parikh images can be computed much more efficiently.

Two-Variable Logic with Counting is Cardinality-Linear. We next explain why the results in [34] imply that C^2 is also cardinality linear. This fact follows by examining the proofs of lemmas 13 and 14 in [34, Page 21]. These lemmas establish a correspondence between solutions of equivalence classes of models of a C^2 formula φ, and certain (singly exponential in formula size) quantifier-free Presburger Arithmetic formulas P. The relevant fact from this construction is that the equivalence classes on models preserve the set of 1-types induced by elements in the model (see the proof of [34, Lemma 10]), and that certain variables (denoted u_i in [34]) in P denote precisely the number of elements realizing each 1-type π_i. Because a 1-type is a conjunction containing every unary predicate or its negation, all elements realizing a given 1-type belong to the same Venn region. The cardinality of a Venn region is therefore a sum of at most singly exponentially many variables u_i. Therefore, the set of all values of cardinalities of sets in models of φ can be obtained as the set of solutions of a Presburger Arithmetic formula. This formula can be obtained by adding summation and existential quantification to the formula P used in the decision procedure for C^2.

We have sketched the result in the context of an expressive and expensive (NEXPTIME-complete) logic C^2. Similar counting techniques are standard for related logics, including description logics [8, Chapter 6].

3.1 Further Remarks

Why QFBAPA has the right expressive power. We have seen that projecting onto set variables yields formulas equivalent to QFBAPA formulas, not only in BAPA but also in WS2S and C^2. A form of converse also holds for the logics we describe: the projections onto set relations of these logics have the expressive power of QFBAPA; meaning that QFBAPA is neither too weak nor too strong to express the projections onto set variables for these logics. Observe that both WS2S and C^2 can define quantifier-free Boolean algebra constraints using e.g. pointwise quantification over elements. However, they can also express more. For C^2, consider the formula

$$(\forall x.\exists^{=1} y.R(x,y)) \wedge (\forall x.\exists^{=1} y.R(y,x)) \wedge (\forall y.\ B(y) \leftrightarrow (\exists x.A(x) \wedge R(x,y)))$$

where R is a fresh binary predicate establishing the bijection between A and B. This constraint expresses $|A| = |B|$ without imposing any additional constraint on A and B. Similar examples can be given for WS2S. Because of such examples, pure Boolean algebra of sets is not sufficient to compute the exact projection.

Purification. Above, we assumed that the shared sets are explicitly given. This is analogous to assuming a purified form of formulas in Nelson-Oppen descriptions. Procedures bases on Nelson-Oppen style combination contain a purification step that introduces fresh variables for subterms of a quantifier-free formula. For combining rich logics that share sets, the purification extracts formulas with one free variable. For example, purification of the formula $|\{x.\, \exists y.\, R(y,x) \wedge \mathit{reach}(\mathit{root}, y)\}| = n$ results in the formula

$$A{=}\{x.\, \mathit{reach}(\mathit{root}, x)\} \wedge B{=}\{x.\, \exists y.\, R(y,x) \wedge y \in A\} \wedge |B| = n$$

Reusing existing decision procedures. If the goal is to obtain a complete decision procedure for combined logics, it is not possible to use the individual decision procedures as a black box (by e.g. enumerating the QFBAPA formulas). This is because there are infinitely many non-equivalent QFBAPA formulas. The projection procedures for WS2S and C^2 can be made as relatively simple modifications of the decision procedures. However, the idea of enumerating a specific target class of QFBAPA formulas is a useful, even if incomplete, way of reusing existing decision procedure implementations for reasoning about the combination of logics. We have implemented one such approach as part of a loop invariant inference algorithm.

4 Sharing Sets using Symbolic Shape Analysis

In order to make our combination result for cardinality-linear logics applicable in practice, we restrict the language used to express the projections onto shared set variables from QFBAPA to a specific class of Boolean algebra formulas. This class of formulas is determined by the abstract domain of a symbolic shape analysis, i.e., the invariants computed by the analysis express properties over the shared sets. The idea behind our approach is to utilize the machinery of the shape analysis to compute consequences over shared sets that communicate information between the different decision procedures.

We define our analysis in terms of abstract interpretation [11]. The concrete domain is given by sets of program states ordered by set inclusion. The concrete fixed point functional is the strongest postcondition operator post, i.e., the analysis computes an over-approximation of the reachable program states or, in other words, an invariant. The abstract domain is given by a lattice of canonical formulae (ordered by logical entailment) and we use decision procedures to automatically compute the abstraction of the concrete post operator. We now briefly explain how the abstract domain looks like and how the abstract post operator is computed. For a detailed description see [38].

Abstract domain. Our abstract domain generalizes the one used in predicate abstraction [16]. It is parameterized by a finite set of *abstraction predicates*. In contrast to predicate abstraction, abstraction predicates do not denote sets of program states but sets of objects in program states (such as objects in the heap). In the following, we fix a particular finite set of set variables $\mathcal{S}$ that serve as our abstraction predicates. We

assume that each $s \in \mathcal{S}$ comes with a defining formula s_d in the form of a set comprehension $\{x.\, F(x)\}$ where $F(x)$ is an HOL formula whose free variables range over x, the program variables, and $\mathcal{S}$. We require that there are no cyclic dependencies between the defining formulas with respect to variables in $\mathcal{S}$. The elements of $\mathcal{S}$ are the shared sets, i.e., each $F(x)$ belongs to one of the combined logics. For notational convenience we assume that $\mathcal{S}$ is closed under complementation of its members.

The abstract domain $AbsDom$ is given by disjunctions of formulas of the form $B(\mathcal{S})=\mathcal{U}$ where $B(\mathcal{S})$ is a set algebraic expression built from variables in $\mathcal{S}$, set union, intersection, and complement. We canonically represent these formulas using BDDs. The meaning of an abstract domain element F is given by the formula $\exists \mathcal{S}.\, (\mathcal{S}{=}\mathcal{S}_d) \wedge F$ where $\mathcal{S}{=}\mathcal{S}_d$ is the conjunction of all $s{=}s_d$ for $s \in \mathcal{S}$.

Abstract post operator. We assume that the concrete program is given by a control flow graph whose edges are labeled by commands from some guarded command language. In each iteration of the analysis the abstract post operator $\mathsf{post}^{\#}$ for some command in the program is applied to the abstract domain element obtained from the previous iteration. We assume that each $F \in AbsDom$ is given in a normal form $\bigvee_i (\bigcup_j \boldsymbol{s}_{i,j}){=}\mathcal{U}$ where the $\boldsymbol{s}_{i,j}$ are Venn regions over $\mathcal{S}$. Then the abstract post for such an F and command cm is defined by:

$$\mathsf{post}^{\#}(cm, F) \stackrel{\text{def}}{=} \bigvee_i \left(\bigcup_j \bigcap \{\, s \in \mathcal{S} \mid\models \boldsymbol{s}_{i,j} \subseteq \mathsf{wlp}^{\#}(cm, F, s) \,\} \right) {=}\mathcal{U} \quad \text{where}$$

$$\mathsf{wlp}^{\#}(cm, F, s) \stackrel{\text{def}}{=} \bigcup \{\, \boldsymbol{s}_1 \mid \mathcal{S}{=}\mathcal{S}_d \wedge F \wedge x \in \boldsymbol{s}_1 \models \mathsf{wlp}(cm, (\mathcal{S}{=}\mathcal{S}_d) \rightarrow x \in s_d) \,\}$$

and where the $\boldsymbol{s}_1$ are Venn regions over sets in $\mathcal{S}$. Here, wlp denotes the weakest liberal precondition operator. We assume that the commands of the program satisfy the condition that the individual logics are closed under computation of wlp. From the above equations follows that we can compute $\mathsf{post}^{\#}(cm, F)$ by deciding satisfiability of formulas of the form

$$\mathcal{S} = \mathcal{S}_d \wedge F \wedge x \in \boldsymbol{s} \wedge \neg \mathsf{wlp}(cm, (\mathcal{S}{=}\mathcal{S}_d) \rightarrow x \in s_d) \tag{3}$$

We use a sound but in general incomplete test for checking satisfiability of such formulas.

Checking satisfiability. Let $upd(cm, s)$ be the formula $\mathsf{wlp}(cm, (\mathcal{S}{=}\mathcal{S}_d) \rightarrow x \in s_d)$. The sets shared in formulas of the form (3) are the sets in $\mathcal{S}$ and sets resulting from the purification of $upd(cm, s)$. Let D_s be the decision procedure for the logic in which s_d is expressed and let $s_1, \ldots, s_n \in \mathcal{S}$ be the sets occuring in s_d. Suppose all s_i have defining formulas expressed in the logic of D_s. Then we can check satisfiability of (3) using D_s only because F already captures consequences of the definitions of sets in $\mathcal{S}$ and $upd(cm, s)$ is equivalent to a formula in the logic of D_s.

If some of the s_i are not defined in the logic of D_s then we need to infer consequences correlating the sets shared in $upd(cm, s)$ and $\mathcal{S}$ using the appropriate decision procedures and propagate these consequences to D_s. Now, if the command cm is deterministic (which is true for most commands in practical programming languages) then the sets shared in $upd(cm, s)$ are exactly the sets whose defining formulas are given by

benchmark	DP	time (in s)	rel. DP time	#sets	#DP calls
SizeList.add	MONA, BAPA	1	99.2%	5	48
SizeList.remove	MONA, BAPA	4	97.8%	9	229
SortedList.add	MONA, Z3	10	97.9%	8	470
SortedList.remove	MONA, Z3	82	99.1%	16	1045
SizeDataTree.add	MONA, Z3, BAPA	386	99.9%	15	452
ThreadedTree.add	MONA, Z3	446	99.7%	17	2882

Fig. 6. Details of evaluation: the columns lists the benchmark name, the used decision procedures (DP), total running time, relative time spent in DPs, number of shared sets used for the abstraction, and the total number of DP calls.

the $upd(cm, s_i)$. We can thus first recursively compute updates $\mathsf{wlp}^{\#}(cm, F, s_i)$ for all sets s_i and then, instead of deciding satisfiability of (3), we use only decision procedure D_s to check satisfiability of the formula

$$\bigwedge_{i\in[0,n]} upd^{\#}(cm, F, s_i) \wedge \mathcal{S}{=}\mathcal{S}_d \wedge F \wedge x \in \boldsymbol{s} \wedge \neg\mathsf{wlp}(cm, (\mathcal{S}{=}\mathcal{S}_d) \rightarrow x \in s_d)$$

where $upd^{\#}(cm, F, s) \stackrel{\text{def}}{=} \left(s \cup (\mathsf{wlp}^{\#}(cm, F, s))^c\right) \cap \left(s^c \cup (\mathsf{wlp}^{\#}(cm, F, s^c))^c\right) {=}\mathcal{U}$

This gives us a sound approximation of the decision procedure described in Section 3. In our implementation we do not check satisfiability of formulas of form (3) for all Venn regions $\boldsymbol{s}$ over $\mathcal{S}$ but only for intersections of sets in $\mathcal{S}$ of a fixed size. This results in a polynomial bound on the number of decision procedure calls that are performed in one iteration of the fixed point computation.

5 Implementation and Evaluation

We integrated a prototype of our method in the Bohne Verifier, the implementation of our symbolic shape analysis. Bohne is implemented on top of the Jahob system. Instead of inferring first the invariants on the shared sets of the target properties and then checking verification conditions generated from the annotated program, our analysis checks the target properties immediately during the fixed point computation (which is wrapped in an abstraction refinement loop [38, Chapter 4]).

We used our implementation to verify properties of data structure operations for data structures such as lists and trees with size and data fields, sorted lists, and sorted threaded trees, including the example from Section 2. The verified properties include 1) absence of runtime errors, preservation of representation invariants such as 2) treeness, 3) sortedness, 4) invariants on size fields, and 5) post conditions expressing the changes on the content of the data structure. Our benchmarks are available on the web[3]. The details of our evaluation are shown in Figure 6. Note that all the shared sets that define the abstraction either result from purification of the target properties or are automatically inferred by the analysis.

[3] http://www.javaverification.org/sets

Some existing tools can handle subsets of the programs and properties we verified. For instance, TVLA has been used to verify functional correctness of operations on trees [35] and sortedness properties on lists [27]. Similar results on lists have been achieved using analyses based on separation logic [28]. The HAVOC tool implements a decision procedure for reasoning about transitive closure and arithmetic properties (including sortedness) [25], but can only handle lists. We are not aware of any tool that can handle all of our examples with the same degree of automation.

6 Conclusions

Many verification techniques rely on decision procedures to achieve a high degree of automation. The class of properties that such techniques are able to verify is therefore limited by the expressive power of the logics supported by the underlying decision procedures. We have presented a combination result for logics that share operations on sets. This result yields an expressive decidable logic that is useful for software verification. We demonstrated the usefulness of this logic by integrating it into our symbolic shape analysis, and verifying properties of linked data structures that no other existing tool can handle with the same degree of automation. We therefore believe that we made an important step in creasing the class of properties that are amenable to automated verification.

References

1. P. Abdulla, A. Annichini, S. Bensalem, A. Bouajjani, P. Habermehl, and Y. Lakhnech. Verification of infinite-state systems by combining abstraction and reachability analysis. In *CAV'99*, 1999.
2. N. Amla, X. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan. An analysis of SAT-based model checking techniques in an industrial environment. In *CHARME*, 2005.
3. T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. In *Proc. ACM PLDI*, 2001.
4. C. Barrett and C. Tinelli. CVC3. In *CAV*, volume 4590 of *LNCS*, 2007.
5. D. Basin and S. Friedrich. Combining WS1S and HOL. In *FROCOS*, volume 7 of *Studies in Logic and Computation*. 2000.
6. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT instead of BDDs. In *Proc. 36th Design Automation Conference*, 1999.
7. R. S. Boyer and J. S. Moore. Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic. In *Machine Intelligence*, volume 11, pages 83–124. 1988.
8. D. Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms*. PhD thesis, Universita di Roma "La Sapienza", 1996.
9. S. Chaki, E. Clarke, A. Groce, S. Jha, and H. Veith. Modular verification of software components in c. In *ICSE 2003*, 2003.
10. E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. SATABS: SAT-based predicate abstraction for ANSI-C. In *TACAS*, volume 3440 of *LNCS*, 2005.
11. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th POPL*, pages 269–282, 1979.
12. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.
13. S. Feferman and R. L. Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.

14. Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In *CADE*, 2007.
15. S. Ginsburg and E. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.
16. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Proc. 9th CAV*, pages 72–83, 1997.
17. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *31st POPL*, 2004.
18. F. Klaedtke and H. Rueß. Monadic second-order logics with cardinalities. In *ICALP*, volume 2719 of *LNCS*, 2003.
19. N. Klarlund and A. Møller. *MONA Version 1.4 User Manual*. BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus, January 2001.
20. S. Krstic, A. Goel, J. Grundy, and C. Tinelli. Combined satisfiability modulo parametric theories. In *TACAS*, pages 602–617, 2007.
21. V. Kuncak. *Modular Data Structure Verification*. PhD thesis, EECS Department, Massachusetts Institute of Technology, February 2007.
22. V. Kuncak, H. H. Nguyen, and M. Rinard. An algorithm for deciding BAPA: Boolean Algebra with Presburger Arithmetic. In *20th International Conference on Automated Deduction, CADE-20*, Tallinn, Estonia, July 2005.
23. V. Kuncak, H. H. Nguyen, and M. Rinard. Deciding Boolean Algebra with Presburger Arithmetic. *J. of Automated Reasoning*, 2006.
24. V. Kuncak and M. Rinard. Towards efficient satisfiability checking for Boolean Algebra with Presburger Arithmetic. In *CADE-21*, 2007.
25. S. K. Lahiri and S. Qadeer. Back to the future: revisiting precise program verification using smt solvers. In *POPL*, 2008.
26. S. K. Lahiri and S. A. Seshia. The UCLID decision procedure. In *CAV'04*, 2004.
27. A. Loginov, T. Reps, and M. Sagiv. Abstraction refinement via inductive learning. In *CAV*, 2005.
28. S. Magill, J. Berdine, E. M. Clarke, and B. Cook. Arithmetic strengthening for shape analysis. In *SAS*, pages 419–436, 2007.
29. S. McLaughlin, C. Barrett, and Y. Ge. Cooperating theorem provers: A case study combining HOL-Light and CVC Lite. In *PDPAR*, volume 144(2) of *ENTCS*, 2006.
30. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM TOPLAS*, 1(2):245–257, 1979.
31. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th CADE*, volume 607 of *LNAI*, pages 748–752, jun 1992.
32. R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.
33. A. Podelski and T. Wies. Boolean heaps. In *SAS'05*, pages 267–282, 2005.
34. I. Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.
35. J. Reineke. Shape analysis of sets. Master's thesis, Universität des Saarlandes, 2005.
36. J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.
37. C. Tinelli and C. Ringeissen. Unions of non-disjoint theories and combinations of satisfiability procedures. *Theor. Comp. Sci.*, 290(1):291–353, Jan. 2003.
38. T. Wies. *Symbolic Shape Analysis*. PhD thesis, University of Freiburg, 2009.
39. T. Wies, V. Kuncak, P. Lam, A. Podelski, and M. Rinard. Field constraint analysis. In *VMCAI*, 2006.
40. C. G. Zarba. A tableau calculus for combining non-disjoint theories. In *TABLEAUX'02*, pages 315–329, 2002.
41. K. Zee, V. Kuncak, and M. Rinard. Full functional verification of linked data structures. In *PLDI*, 2008.