Building a Calculus of Data Structures

Viktor Kuncak[1⋆], Ruzica Piskac[1], Philippe Suter[1], and Thomas Wies[2]

[1] EPFL School of Computer and Communication Sciences, Lausanne, Switzerland
firstname.lastname@epfl.ch
[2] Institute of Science and Technology Austria, Klosterneuburg, Austria
wies@ist.ac.at

Abstract. Techniques such as verification condition generation, predicate abstraction, and expressive type systems reduce software verification to proving formulas in expressive logics. Programs and their specifications often make use of data structures such as sets, multisets, algebraic data types, or graphs. Consequently, formulas generated from verification also involve such data structures. To automate the proofs of such formulas we propose a logic (a "calculus") of such data structures. We build the calculus by starting from decidable logics of individual data structures, and connecting them through functions and sets, in ways that go beyond the frameworks such as Nelson-Oppen. The result are new decidable logics that can simultaneously specify properties of different kinds of data structures and overcome the limitations of the individual logics.
Several of our decidable logics include abstraction functions that map a data structure into its more abstract view (a tree into a multiset, a multiset into a set), into a numerical quantity (the size or the height), or into the truth value of a candidate data structure invariant (sortedness, or the heap property). For algebraic data types, we identify an asymptotic many-to-one condition on the abstraction function that guarantees the existence of a decision procedure.
In addition to the combination based on abstraction functions, we can combine multiple data structure theories if they all reduce to the same data structure logic. As an instance of this approach, we describe a decidable logic whose formulas are propositional combinations of formulas in: weak monadic second-order logic of two successors, two-variable logic with counting, multiset algebra with Presburger arithmetic, the Bernays-Schönfinkel-Ramsey class of first-order logic, and the logic of algebraic data types with the set content function. The subformulas in this combination can share common variables that refer to sets of objects along with the common set algebra operations. Such sound and complete combination is possible because the relations on sets definable in the component logics are all expressible in Boolean Algebra with Presburger Arithmetic. Presburger arithmetic and its new extensions play an important role in our decidability results. In several cases, when we combine logics that belong to NP, we can prove the satisfiability for the combined logic is still in NP.

⋆ This research is supported in part by the Swiss National Science Foundation Grant #120433 "Precise and Scalable Analyses for Reliable Software".

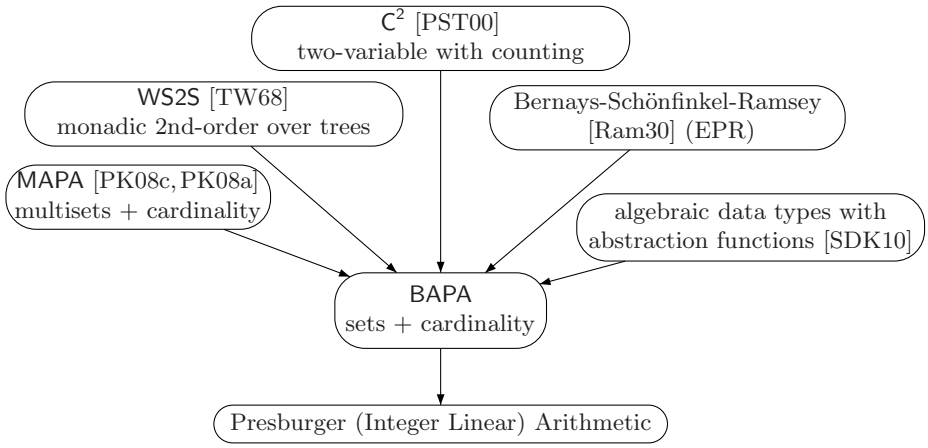

Fig. 1. Components of our decidable logic and reductions used to show its decidability.

1 Introduction

A cornerstone of software verification is the problem of proving the validity of logical formulas that describe software correctness properties. Among the most effective tools for this task are the systems (e.g. [dMB08]) that incorporate decision procedures for data types that commonly occur in software (e.g. numbers, sets, arrays, algebraic data types). Such decision procedures leverage insights into the structure of these data types to reduce the amount of uninformed search they need to perform. Prominent examples of decision procedures are concerned with numbers, including (the quantifier-free fragment of) Presburger arithmetic (PA) [Pre29]. For the verification of modern software, data structures are at least as important as numerical constraints. Among the best behaved data structures are sets, with Boolean algebra of sets [Sko19] among the basic decidable examples, others include algebraic data types [Opp78,BST07] and arrays [SBDL01,BM07,dMB09]. Reasoning about imperative data structures can be described using formulas interpreted over graphs; decidable fragments of first-order logic present a good starting point for such reasoning [BGG97].

In this paper we give an overview of some recent decision procedures for reasoning about data structures, including sets and multisets with cardinality bounds, algebraic data types with abstraction functions, and combinations of expressive logics over trees and graphs. Our results illustrate the rich structure of connections between logics of different data structures and numerical constraints. Figure 1 illustrates some of these connections; they present combinations that go beyond the disjoint combination framework of Nelson-Oppen [NO79].

Given logics A and B we often consider a combined logic $c(A, B)$ that subsumes A and B and has additional operators that make the combined logic more useful (e.g. abstraction functions from A to B, or numerical measures of the data structure). In such situation, we have found it effective to reduce the combina-

tion $c(A, B)$ to one of the logic, say B. Under certain conditions, if we consider another combination $c'(A', B)$ we can obtain the decidability of the combination $c''(A, A', B)$ of all three logics. When B is the propositional logic, this idea has been applied to combine logics that share only equality (e.g. [LS04]).

In our approach, we take as the base logic B a logic of sets with cardinality operator, which we call *Boolean Algebra with Presburger Arithmetic (BAPA)*. BAPA is much richer than propositional logic. Consequently, we can use BAPA to combine logics that share not only equalities but also sets of objects. Different logics define the sets in different ways: first-order logic fragments use unary predicates to define sets, other logics have variables denoting sets (this includes monadic second-order logic, the logics of multisets, and the logic of algebraic data types with abstractions). A key technical challenge is establishing reductions from new logics to existing ones, and casting known decision procedures as reductions to BAPA. The formulas in our combined logic are quantifier-free combinations of possibly quantified formulas of component logics. Our approach leads to the decidability of classes of complex verification conditions for which we previously had only heuristic, incomplete, approaches.

The results we present follow [KR07,PK08a,PK08c,PK08b,SDK10,WPK09].

2 Boolean Algebra with Presburger Arithmetic

We start by considering a logic that combines two well-known decidable logics: 1) the algebra of sets (with operations such as union, intersection, complement, and relations such as extensional set equality and subset), and 2) Presburger arithmetic [Pre29] (with linear arithmetic expressions over integer variables). We establish a connection between these two logics by introducing the cardinality operator that computes the number of elements in the set expression. We call this logic BAPA (Boolean Algebra with Presburger Arithmetic) [KNR06], and focus on its quantifier-free fragment (QFBAPA). Figure 2 shows the syntax of QFBAPA. Figure 3 shows example verification conditions that it can express.

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

Fig. 2. Quantifier-Free Boolean Algebra with Presburger Arithmetic (QFBAPA)

Methods to decide QFBAPA. Like Presburger arithmetic [Pre29], BAPA admits quantifier elimination [KNR06], which gives NEXPTIME decision procedure for quantifier-free formulas. The logic also admits small model property, but, due to formulas such as $|A_0| = 1 \wedge \bigwedge_i |A_i| = 2|A_{i-1}|$, the number of assignments to

verification condition	property being checked
$x \notin \texttt{content} \wedge \texttt{size} = \texttt{card content} \longrightarrow$ $(\texttt{size} = 0 \leftrightarrow \texttt{content} = \emptyset)$	using invariant on size to prove correctness of an efficient emptiness check
$x \notin \texttt{content} \wedge \texttt{size} = \texttt{card content} \longrightarrow$ $\texttt{size} + 1 = \texttt{card}(\{x\} \cup \texttt{content})$	maintaining correct size when inserting fresh element
$\texttt{size} = \texttt{card content} \wedge$ $\texttt{size1} = \texttt{card}(\{x\} \cup \texttt{content}) \longrightarrow$ $\texttt{size1} \leq \texttt{size} + 1$	maintaining size after inserting any element
$\texttt{content} \subseteq \texttt{alloc} \wedge$ $x_1 \notin \texttt{alloc} \wedge$ $x_2 \notin \texttt{alloc} \cup \{x_1\} \wedge$ $x_3 \notin \texttt{alloc} \cup \{x_1\} \cup \{x_2\} \longrightarrow$ $\texttt{card}\,(\texttt{content} \cup \{x_1\} \cup \{x_2\} \cup \{x_3\}) =$ $\texttt{card}\,\texttt{content} + 3$	allocating and inserting three objects into a container data structure
$x \in C \wedge C_1 = (C \setminus \{x\}) \wedge$ $\texttt{card}(\texttt{alloc1} \setminus \texttt{alloc0}) \leq 1 \wedge$ $\texttt{card}(\texttt{alloc2} \setminus \texttt{alloc1}) \leq \texttt{card}\,C_1 \longrightarrow$ $\texttt{card}\,(\texttt{alloc2} \setminus \texttt{alloc0}) \leq \texttt{card}\,C$	bound on the number of allocated objects in a recursive function that incorporates container C into another container

Fig. 3. Example verification conditions that belong to QFBAPA

set variables can be doubly exponential, which would again give NEXPTIME procedure. We can obtain an optimal worst-case time decision procedure for QFBAPA using two insights. The first insight follows the quantifier elimination algorithm and introduces an integer variable for each Venn region (an intersection of set variables and their complements), reducing the formula to Presburger arithmetic. The second insight shows that the generated Presburger arithmetic formulas enjoy a *sparse model* property: if they are satisfiable, they are satisfiable in a model where most variables (denoting sizes of Venn regions) are zero. Using an appropriate encoding it is possible to generate a polynomial-sized instead of exponential-sized Presburger arithmetic formula. This can be used to show that the satisfiability problem for QFBAPA remains within NP [KR07].

3 Multisets Algebra with Presburger Arithmetic

The decidability and NP completeness of QFBAPA also extends to Multiset Algebra with Presburger Arithmetic, in which variables can denote both sets and multisets (and where one can test whether a set is a multiset, or convert a multiset into a set). The motivation for multisets comes from verification of data structures with possibly repeated elements, where, in addition to knowing whether an element occurs in the data structure, we are also interested how many times it occurs. A detailed description of decision procedures for satisfiability of multisets with cardinality constraints is in [PK08a, PK08c, PK08b].

A multiset is a function m from a fixed finite set E to $\mathbb{N}$, where $m(e)$ denotes the number of times an element e occurs in the multiset (multiplicity of e). In addition to multiset operations such as multiplicity-preserving union and the intersection, every PA formula defining a relation leads to an operation on multisets in our logic, defined point-wise using this relation. For example, $(m_1 \cap m_2)(e) = \min(m_1(e), m_2(e))$ and $m_1 \subseteq m_2$ means $\forall e.m_1(e) \leq m_2(e)$. The logic also supports the cardinality operator that returns the number of elements in a multiset. The cardinality operator is a useful in applications, yet it prevents the use of previous decision procedures for arrays [BM07] to decide our logic. Figure 4 summarizes the language of multisets with cardinality constraints. There are two levels at which integer linear arithmetic constraints occur: to define point-wise operations on multisets (inner formulas) and to define constraints on cardinalities of multisets (outer formulas). Integer variables from outer formulas cannot occur within inner formulas.

Top-level formulas:

$$F ::= A \mid F \wedge F \mid \neg F$$
$$A ::= M{=}M \mid M \subseteq M \mid \forall e.\mathsf{F}^{\mathsf{in}} \mid \mathsf{A}^{\mathsf{out}}$$

Outer linear arithmetic formulas:

$$\mathsf{F}^{\mathsf{out}} ::= \mathsf{A}^{\mathsf{out}} \mid \mathsf{F}^{\mathsf{out}} \wedge \mathsf{F}^{\mathsf{out}} \mid \neg \mathsf{F}^{\mathsf{out}}$$
$$\mathsf{A}^{\mathsf{out}} ::= \mathsf{t}^{\mathsf{out}} \leq \mathsf{t}^{\mathsf{out}} \mid \mathsf{t}^{\mathsf{out}}{=}\mathsf{t}^{\mathsf{out}} \mid (\mathsf{t}^{\mathsf{out}}, \ldots, \mathsf{t}^{\mathsf{out}}){=}\sum_{\mathsf{F}^{\mathsf{in}}}(\mathsf{t}^{\mathsf{in}}, \ldots, \mathsf{t}^{\mathsf{in}})$$
$$\mathsf{t}^{\mathsf{out}} ::= k \mid \ \mid M \mid \ \mid C \mid \mathsf{t}^{\mathsf{out}} + \mathsf{t}^{\mathsf{out}} \mid C \cdot \mathsf{t}^{\mathsf{out}} \mid \text{if } \mathsf{F}^{\mathsf{out}} \text{ then } \mathsf{t}^{\mathsf{out}} \text{ else } \mathsf{t}^{\mathsf{out}}$$

Inner linear arithmetic formulas:

$$\mathsf{F}^{\mathsf{in}} ::= \mathsf{A}^{\mathsf{in}} \mid \mathsf{F}^{\mathsf{in}} \wedge \mathsf{F}^{\mathsf{in}} \mid \neg \mathsf{F}^{\mathsf{in}}$$
$$\mathsf{A}^{\mathsf{in}} ::= \mathsf{t}^{\mathsf{in}} \leq \mathsf{t}^{\mathsf{in}} \mid \mathsf{t}^{\mathsf{in}}{=}\mathsf{t}^{\mathsf{in}}$$
$$\mathsf{t}^{\mathsf{in}} ::= m(e) \mid C \mid \mathsf{t}^{\mathsf{in}} + \mathsf{t}^{\mathsf{in}} \mid C \cdot \mathsf{t}^{\mathsf{in}} \mid \text{if } \mathsf{F}^{\mathsf{in}} \text{ then } \mathsf{t}^{\mathsf{in}} \text{ else } \mathsf{t}^{\mathsf{in}}$$

Multiset expressions:

$$M ::= m \mid \emptyset \mid M \cap M \mid M \cup M \mid M \uplus M \mid M \setminus M \mid M \setminus\setminus M \mid \mathsf{set}(M)$$

Terminals:

m - multiset variables; e - index variable (fixed)
k - integer variable; C - integer constant

Fig. 4. Quantifier-Free Multiset Constraints with Cardinality Operator

First, we sketch the decision procedure from [PK08a]. Given a formula F_M, we convert it to the following sum normal form:

$$P \ \wedge \ (u_1, \ldots, u_n) = \sum_{e \in E}(t_1, \ldots, t_n) \wedge \forall e.F$$

where

- P is a quantifier-free PA formula without any multiset variables
- the variables in $t_1, \ldots, t_n$ and F occur only as expressions of the form $m(e)$ for m a multiset variable and e the fixed index variable
- formula P can only share variables with terms $u_1, \ldots, u_n$.

The algorithm that reduces a formula to its sum normal form runs in polynomial time. The goal of our decision procedure is to express the subformula $(u_1, \ldots, u_n) = \sum_{e \in E} (t_1, \ldots, t_n) \wedge \forall e.F$ as a quantifier-free PA formula and thus reduce the satisfiability of a formula belonging to the language of Figure 4 to satisfiability of quantifier-free PA formulas. As a first step, we use the fact that a formula in the sum normal form

$$P \ \wedge \ (u_1, \ldots, u_n) = \sum_{e \in E} (t_1, \ldots, t_n) \ \wedge \ \forall e.F$$

is equisatisfiable with the formula

$$P \ \wedge \ (u_1, \ldots, u_n) \in \{(t'_1, \ldots, t'_n) \mid F'\}^*$$

where the terms t'_i and the formula F' are formed from the terms t_i and the formula F in the following way: for each multiset expression $m_i(e)$ we introduce a fresh new integer variable x_i and then we substitute each occurrence of $m_j(e)$ in the terms terms t_i and the formula F with the corresponding variable x_j. The star-closure of a set C is defined as $C^* = \{\boldsymbol{v}_1 + \ldots + \boldsymbol{v}_n \mid \boldsymbol{v}_1, \ldots \boldsymbol{v}_n \in C \wedge n \geq 0\}$. We are left with the problem of deciding the satisfiability of quantifier-free PA formulas extended with the star operator [PK08c]. For this we need representations of solutions of PA formulas using semilinear sets.

3.1 Semilinear Sets

Let $S \subseteq \mathbb{N}^n$ be a set of vectors of non-negative integers and let $a \in \mathbb{N}^n$ be a vector of non-negative integers. A linear set $LS(a; S)$ is defined as $LS(a; S) = \{a + x_1 + \ldots + x_n \mid x_i \in S \wedge n \geq 0\}$. A vector a is called the *base* vector, while elements of S are called the *step* vectors. A semilinear set Z is defined as a finite union of linear sets: $Z = \cup_{i=1}^{n} LS(a_i; S_i)$.

By definition, a semilinear set can be described as a solution set of a PA formula. [GS66] showed that the converse also holds: the solution of a PA formula is a semilinear set.

Consider the set $\{(t'_1, \ldots, t'_n) \mid F'\}^*$. The set of all vectors which are solution of formula F' is a semilinear set. Moreover, it is not difficult to see that applying the star operator on a semiliner set results with the set which can be described with the Presburger arithmetic formula. Consequently, applying the star operator on a semiliner set results in a new semilinear set. Because $\{(t'_1, \ldots, t'_n) \mid F'\}^*$ is a semilinear set, checking whether $(u_1, \ldots, u_n) \in \{(t'_1, \ldots, t'_n) \mid F'\}^*$ is effectively expressible as a Presburger arithmetic formula. Consequently, satisfiability of an initial multiset constraints problem reduces to satisfiability of quantifier-free Presburger arithmetic formulas. Following the constructions behind these closure properties gives the decidability, but, unfortunately, not the optimal NP complexity.

3.2 NP Complexity of Multisets with Cardinality Constraints

To show NP membership of the language in Figure 4, we prove the linear arithmetic with positively occurring stars is in NP [PK08c]. We use theorems bounding the sizes of semilinear sets and again apply a sparse model theorem.

Bounds on Size of the Vectors Defining Semilinear Set. In [Pot91] Pottier investigates algorithms and bounds for solving a system of integer constraints. The algorithm presented there runs in singly exponential time and returns a semilinear set Z which is a solution of the given system. This paper also establishes the bounds on the size of base and step vectors which occurs in the definition of Z. Let $x = (x_1, \ldots, x_n)$ be an integer vector. We use two standard norms of the vector x:

- $||x||_1 = \sum_{i=1}^{n} |x_i|$
- $||x||_\infty = \max_{i=1}^{n} |x_i|$

For matrices we use the norm $||A||_{1,\infty} = sup_i\{\sum_j |a_{ij}|\}$.

Fact 1 ([Pot91], Corollary 1) *Given a system $Ax \leq b$, a semilinear set describing the solution can be computed in singly exponential time. Moreover, if $\boldsymbol{v}$ is a base or a step vector that occurs in the semilinear set, then*

$$||v||_1 \leq (2 + ||A||_{1,\infty} + ||b||_\infty)^m$$

Let F be a Presburger arithmetic formula and let Z be a semilinear set describing a set of solutions of F. Let S be a set of all base and step vectors of Z. Theorem 1 implies that there exists polynomial $p(s)$, where s is a size of the input formula, such that for each $v \in S$, $||v||_1 \leq 2^{p(s)}$.

Sparse Solutions of Integer Cones. Let S be a set of integer vectors. For a vector $v \in S^*$ we are interested in the minimal number of vectors from S such that v is their linear combination. Eisenbrand and Shmonin [ES06] proved that this minimal number depends only on the dimension of vectors and on the size of the coefficients of those vectors, as follows.

Fact 2 ([ES06], Theorem 1(ii)) *Let S be a finite set of integer vectors of the dimension n and let $v \in S^*$. Then there exists $S_1 \subseteq S$ such that $b \in {S_1}^*$ and $|S_1| \leq 2n \log(4nM)$, where $M = \max_{x \in S} ||x||_\infty$.*

Small Model Property for Integer Linear Programming. [Pap81] proves the small model property for systems of integer linear constrains $A\boldsymbol{x} = \boldsymbol{b}$.

Fact 3 ([Pap81]) *Given an $m \times n$ integer matrix A, an m-dimensional integer vector $\boldsymbol{b}$ and an integer M such that $||A||_{1,\infty} \leq M$ and $||\boldsymbol{b}||_\infty \leq M$, if the system $A\boldsymbol{x} = \boldsymbol{b}$ has a solution, then it also has a non-negative solution vector $\boldsymbol{v}$ such that $||\boldsymbol{v}||_\infty \leq n(mM)^{2m+1}$.*

Membership in NP. Back to our formula F', consider a semilinear set $Z = \cup_{i=1}^{k} LS(a_i; \{b_{i1}, \ldots, b_{ik_i}\})$ which corresponds to the set $\{\boldsymbol{t} \mid F'(\boldsymbol{t})\}$. Elimination of the star operator from the expression $\boldsymbol{u} \in \{\boldsymbol{t} \mid F'(\boldsymbol{t})\}^*$ results in the formula $\boldsymbol{u} = F_N(a_i, b_{ij})$, where F_N is a new Presburger arithmetic formula which has base vectors and step vectors of Z as variables. Using Theorem 2 we can show that there exists equisatisfiable formula $\boldsymbol{u} = F'_N(a_i, b_{ij})$ which uses only polynomially many ($O(n^2 \log n)$) vectors a_i and b_{ij}. The next problem is to verify in polynomial time whether a vector belongs to a set of vectors defining the semilinear set. We showed in [PK08c] that instead of guessing a_i and b_{ij}, it is enough to guess vectors v_c which are solutions of F'.

Using this we proved that $\boldsymbol{u} \in \{\boldsymbol{t} \mid F'(\boldsymbol{t})\}^*$ is equisatisfiable with the formula

$$\boldsymbol{u} = \sum_{i=1}^{Q} \lambda_i \boldsymbol{v}_i \wedge \bigwedge_{i=1}^{Q} F(\boldsymbol{v}_i)$$

where Q is a number (not a variable!) which can be computed from the proofs of the above theorems, and depends on

- dimension of a problem
- $||\cdot||_\infty$ of generating vectors of the semilinear sets

The important is that we do not actually need to compute vectors generating the semilinear set. We only require their norm $||\cdot||_\infty$ and it can be easily calculated by applying Theorem 1.

The last hurdle is that the derived formula does not seem to be linear as it contains multiplication of variables: $\lambda_i \boldsymbol{v}_i$. This problem is solved by applying Theorem 3. Because we know that there exists a bounded solution, we can calculate the concrete bound on the size of the solution and obtain the number r. Using this number we can rewrite v_i as a binary number and expand multiplication this way:

$$\lambda_i v_i = (\sum_{j=0}^{r} v_{ij} 2^j)\lambda_i = \sum_{j=0}^{r} 2^j (v_{ij}\lambda_i) = \sum_{j=0}^{r} 2^j \mathsf{ite}(v_{ij}, \lambda_i, 0) =$$
$$\mathsf{ite}(v_{i0}, \lambda_i, 0) + 2(\mathsf{ite}(v_{i1}, \lambda_i, 0) + 2(\mathsf{ite}(v_{i2}, \lambda_i, 0) + \cdots)))$$

This way we derive the linear arithmetic formula which polynomial in the size of the initial problem and obtain NP-completeness.

4 Algebraic Data Types with Abstraction Functions

In this section, we give an overview of a decision procedure for a logic which combines algebraic data types with an abstraction function mapping these types to elements of a collection theory. The full account of our results is available in [SDK10]. To simplify the presentation, we restrict ourselves to the data type of binary trees storing elements of a countably infinite type, which in Scala [OSV08] syntax would be written as

```
abstract class Tree
case class Node(left: Tree, value: 𝓔, right: Tree) extends Tree
case class Leaf() extends Tree
```

for an element type $\mathcal{E}$. We consider abstraction functions which are given as a catamorphism (generalized fold) over the trees, given as

```
def α(t: Tree): 𝓒 = t match {
  case Leaf() ⇒ empty
  case Node(l,e,r) ⇒ combine(α(l), e, α(r))
}
```

for some functions $\mathsf{empty} : \mathcal{C}$ and $\mathsf{combine} : (\mathcal{C}, \mathcal{E}, \mathcal{C})$. Formally, our logic is parametrized by a collection theory $\mathcal{L}_\mathcal{C}$ and an abstraction function α given in terms of empty and $\mathsf{combine}$ as above. We denote the logic by $\mathcal{T}_\alpha$ (note that $\mathcal{L}_\mathcal{C}$ is implicit in α). Fig. 5 shows the syntax of $\mathcal{T}_\alpha$, and Fig. 6 its semantics. The description refers to the catamorphism α, as well as the semantics of the theory $\mathcal{L}_\mathcal{C}$, denoted $[\![\quad]\!]_\mathcal{C}$.

$$
\begin{array}{rl}
T ::= & t \mid \mathsf{Leaf} \mid \mathsf{Node}(T, E, T) \mid \mathsf{left}(T) \mid \mathsf{right}(T) \qquad \text{Tree terms} \\
C ::= & c \mid \alpha(t) \mid \mathfrak{T}_\mathcal{C} \qquad \mathcal{C}\text{-terms} \\
F_T ::= & T = T \mid T \neq T \qquad \text{Equations over trees} \\
F_C ::= & C = C \mid \mathfrak{F}_\mathcal{C} \qquad \text{Formulas of } \mathcal{L}_\mathcal{C} \\
E ::= & e \qquad \text{Variables of type } \mathcal{E} \\
\phi ::= & \bigwedge F_T \wedge \bigwedge F_C \qquad \text{Conjunctions} \\
\psi ::= & \phi \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \Rightarrow \phi \mid \phi \Leftrightarrow \phi \qquad \text{Formulas}
\end{array}
$$

$\mathfrak{T}_\mathcal{C}$ and $\mathfrak{F}_\mathcal{C}$ represent terms and formulas of $\mathcal{L}_\mathcal{C}$ respectively. Formulas are assumed to be closed under negation.

Fig. 5. Syntax of $\mathcal{T}_\alpha$

$$
\begin{aligned}
[\![\mathsf{Node}(T_1, e, T_2)]\!] &= \mathsf{Node}([\![T_1]\!], [\![e]\!]_\mathcal{C}, [\![T_2]\!]) \\
[\![\mathsf{Leaf}]\!] &= \mathsf{Leaf} \\
[\![\mathsf{left}(\mathsf{Node}(T_1, e, T_2))]\!] &= [\![T_1]\!] \\
[\![\mathsf{right}(\mathsf{Node}(T_1, e, T_2))]\!] &= [\![T_2]\!] \\
[\![T_1 = T_2]\!] &= [\![T_1]\!] = [\![T_2]\!] \\
[\![T_1 \neq T_2]\!] &= [\![T_1]\!] \neq [\![T_2]\!] \\
[\![\alpha(\mathsf{Leaf})]\!] &= [\![\mathsf{empty}]\!]_\mathcal{C} \\
[\![\alpha(\mathsf{Node}([\![T_1]\!], [\![e]\!], [\![T_2]\!]))]\!] &= [\![\mathsf{combine}([\![T_1]\!], [\![e]\!], [\![T_2]\!])]\!]_\mathcal{C} \\
[\![C_1 = C_2]\!] &= [\![C_1]\!]_\mathcal{C} = [\![C_2]\!]_\mathcal{C} \\
[\![\mathfrak{F}_\mathcal{C}]\!] &= [\![\mathfrak{F}_\mathcal{C}]\!]_\mathcal{C} \\
[\![\neg\phi]\!] &= \neg[\![\phi]\!] \\
[\![\phi_1 \star \phi_2]\!] &= [\![\phi_1]\!] \star [\![\phi_2]\!] \text{ where } \star \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}
\end{aligned}
$$

Fig. 6. Semantics of $\mathcal{T}_\alpha$

```
object BSTSet {
  type E = Int
  type C = Set[E]
  abstract class Tree
  case class Leaf() extends Tree
  case class Node(left: Tree, value: E, right: Tree) extends Tree

  // abstraction function α
  def content(t: Tree): C = t match {
    case Leaf() ⇒ Set.empty
    case Node(l,e,r) ⇒ content(l) ++ Set(e) ++ content(r)
  }

  // adds an element to a set
  def add(e: E, t: Tree): Tree = (t match {
    case Leaf() ⇒ Node(Leaf(), e, Leaf())
    case t @ Node(l,v,r) ⇒
      if (e < v) Node(add(e, l), v, r) else if (e == v) t else Node(l, v, add(e, r))
  }) ensuring (res ⇒ content(res) == content(t) ++ Set(e))
}
```

Fig. 7. A binary search tree implementation of a set

4.1 Examples of Applications

A typical target application for our decision procedure is the verification of functional code. Fig. 7 presents an annotated code fragment of an implementation of a set data structure using a binary search tree.[3] Note that the abstraction function content is the catamorphism α defined by $\mathsf{empty} = \emptyset$ and $\mathsf{combine}(t_1, e, t_2) = \alpha(t_1) \cup \{e\} \cup \alpha(t_2)$, and that it is used within the postcondition of the function add. Such specifications in term of the abstraction function are natural because they concisely express the algebraic laws one expects to hold for the data structure.

By applying standard techniques to replace the recursive call in add by the function contract, we obtain (among others) the following verification condition:

$$\forall t_1, t_2, t_3, t_4 : \mathsf{Tree},\ e_1, e_2 : \mathsf{Int}\ .\ t_1 = \mathsf{Node}(t_2, e_1, t_3) \Rightarrow \\ \alpha(t_4) = \alpha(t_2) \cup \{e_2\} \Rightarrow \alpha(\mathsf{Node}(t_4, e_1, t_3)) = \alpha(t_1) \cup \{e_2\}$$

This formula combines constraints over tree terms and over terms in the collection theory (in this case, over sets), as well as a non-trivial connection given by α (the content function in the code).

Other abstraction functions of interest include the computation of a multiset preserving multiplicities instead of a set, the computation of a list of the elements read in, for instance, in-order traversal, the computation of minimal or maximal elements, etc. In fact, even invariants like sortedness of a tree can be expressed as a catamorphism, as shown in Fig 8.

[3] Set.empty, ++ and Set(e) are Scala notations for $\emptyset$, $\cup$ and $\{e\}$ respectively.

```
def sorted(t: Tree): (Option[Int],Option[Int],Boolean) = t match {
  case Leaf() ⇒ (None, None, true)
  case Node(l, v, r) ⇒ {
    (sorted(l),sorted(r)) match {
      case ((_,_,false),_) ⇒ (None, None, false)
      case (_,(_,_,false)) ⇒ (None, None, false)
      case ((None,None,_),(None,None,_)) ⇒ (Some(v), Some(v), true)
      case ((Some(minL),Some(maxL),_),(None,None,_))
        if (maxL < v) ⇒ (Some(minL),Some(v),true)
      case ((None,None,_),(Some(minR),Some(maxR),_))
        if (minR > v) ⇒ (Some(v), Some(maxR), true)
      case ((Some(minL),Some(maxL),_), (Some(minR),Some(maxR),_))
        if (maxL < v && minR > v) ⇒ (Some(minL),Some(maxR),true)
      case _ ⇒ (None,None,false)
}}}
```

Fig. 8. A catamorphism which computes a triple where the first and second elements are the minimal and maximal values of the tree, respectively, and the third is a boolean value indicating whether the tree is sorted.

4.2 The Decision Procedure

We give an overview of our decision procedure for conjunctions of literals of $\mathcal{T}_\alpha$. To lift it to formulas of arbitrary boolean structure, one can follow the DPLL(T) approach [GHN+04].

The general idea of the decision procedure is to first use unification to solve the constraints on the trees, then to derive and propagate all consequences relevant to the type $\mathcal{C}$ of collections that abstracts the trees. In such manner it reduces a problem over trees and their abstract values in $\mathcal{L}_\mathcal{C}$ to a problem in $\mathcal{L}_\mathcal{C}$. We assume a decision procedure is available for $\mathcal{L}_\mathcal{C}$. Instances of such procedures for sets and multisets were presented in sections 2 and 3, for example.

Rewriting into Normal Form. The first steps of the decision procedure consist in rewriting the problem in a normal form more suitable for the final reduction. To this end we:

- separate the equations and disequations between tree terms from the literals of $\mathcal{L}_\mathcal{C}$ by introducing fresh variables and new equalities of the form $c = \alpha(t)$, where c and t are variables representing a collection and a tree respectively (purification)
- flatten the tree terms by introducing fresh variables to represent the subtrees
- eliminate the selector functions (left and right in Fig. 5)

We then guess an arrangement over all tree variables, as well as over the variables denoting elements stored in the nodes of the trees. (Note that this is a non-deterministic polynomial process.) We add to the formula all the equalities and disequalities that represent this arrangement. We then apply unification on the equalities over tree variables and terms. At this point, we either detect unsatisfiability, or we obtain a solved form for the unified equalities. In this solved

form, some tree variables are expressed as being terms built using the Node constructor, the Leaf constant and some other tree variables. We call all variables appearing in such a construction *parameter variables*. A property of unification is that parameter variables are never themselves defined as a term constructed over other variables.

As a final transformation step, we rewrite all terms of the form $\alpha(t)$ where t is a non-parameter tree variable as follows: we replace t by its definition in terms of parameter tree variables from the solved form, and partially evaluate α over this term, using the combine and empty functions which define α. After applying this rewriting everywhere, α is only applied to parameter tree variables, and we can write our formula in the following normal form:

$$N(\boldsymbol{T}(\boldsymbol{t}), \boldsymbol{t}) \wedge M(\boldsymbol{t}, \boldsymbol{c}) \wedge F_E \wedge F_C$$

where:

- $\boldsymbol{t}$ denotes all parameter tree variables
- $\boldsymbol{T}(\boldsymbol{t})$ denotes the terms mapped to the non-parameter variables in the solved form
- $N(\boldsymbol{T}(\boldsymbol{t}))$ is a formula expressing that all parameter variables are distinct, that none of them is equal to Leaf, and that they are all distinct from the terms $\boldsymbol{T}(\boldsymbol{t})$
- $M(\boldsymbol{t}, \boldsymbol{c})$ is a conjunction containing for each parameter variable t_i the conjunct $c_i = \alpha(t_i)$ (c_i is introduced if needed)
- F_E is a conjunction of literals of the form $e_i = e_j$ or $e_i \neq e_j$ expressing the arrangement we guessed over the element variables
- F_C is a formula of $\mathcal{L}_\mathcal{C}$

The formulas F^E and F^C are already expressed in the collection theory. We call D the conjunction $N(\boldsymbol{T}(\boldsymbol{t}), \boldsymbol{t}) \wedge M(\boldsymbol{t}, \boldsymbol{c})$. To ensure the completeness of our decision procedure, we need to find a formula D_M entirely expressed in $\mathcal{L}_\mathcal{C}$ which is equisatisfiable with D. We can then reduce the problem to the satisfiability of $D_M \wedge F_E \wedge F_C$, which we can solve with a decision procedure for $\mathcal{L}_\mathcal{C}$. Note that if we choose a formula D_M which is weaker than D, our decision procedure is still sound, but the equisatisfiability is required for completeness. We now give a sufficient criterion for the existence of such an equisatisfiable formula D_M.

4.3 A Completeness Criterion

In [SDK10], we present two sufficient criteria for obtaining a complete decision procedure. Since the first one is strictly subsumed by the second, we omit it here.

Definition 1 (Tree Shape). *Let* SLeaf *be a new constant symbol and* $\mathsf{SNode}(t_1, t_2)$ *a new constructor symbol. The* shape *of a tree t, denoted $\check{\mathsf{s}}(t)$, is a ground term built from* SLeaf *and* $\mathsf{SNode}(\_, \_)$ *as follows:*

$$\check{\mathsf{s}}(\mathsf{Leaf}) = \mathsf{SLeaf}$$
$$\check{\mathsf{s}}(\mathsf{Node}(T_1, e, T_2)) = \mathsf{SNode}(\check{\mathsf{s}}(T_1), \check{\mathsf{s}}(T_2))$$

Definition 2 (Sufficient Surjectivity). *We call an abstraction function* sufficiently surjective *if and only if, for each natural number $p > 0$ there exist, computable as a function of p*

- *a finite set of shapes S_p*
- *a closed formula M_p in the collection theory such that $M_p(c)$ implies $|\alpha^{-1}(c)| > p$*

such that, for every term t, $M_p(\alpha(t))$ or $\breve{s}(t) \in S_p$.

In practice, the formula M_p can introduce new variables as long as it is existentially closed and the decision procedure for the collection theory can handle positive occurrences of existential quantifiers.

We give in [SDK10] a construction for D_M for any sufficiently surjective abstraction. The intuition behind it is that we can proceed by case analysis on the shapes of the parameter tree variables. Since there are finitely many shapes in S_p, we can encode in our formula D_M all possible assignments of these shapes to the tree variables. The situation where the assigned tree is not of a known shape is handled by adding the condition $M_p(\alpha(t))$, which is then guaranteed to hold by hypotheses on S_p and M_p using a strengthened version of the "independence of disequations lemma" [CD94, Page 178]. We omit the technical details, but the sufficient surjectivity condition implies that for n trees such that $M_p(t_1) \wedge \ldots \wedge M_p(t_n)$ and $\alpha(t_1) = \ldots = \alpha(t_n)$, we can always find assignments to $t_1,\ldots,t_n$ such that p disequalities between them are satisfied (see [SDK10, Section 5.3]). By setting p in our formula to the number of disequalities in $N(\boldsymbol{T}(\boldsymbol{t}), \boldsymbol{t})$ we obtain a formula equisatisfiable with D: since D_M encodes all possible assignments of trees to the variables, D_M is satisfiable if D is. In the other direction, if D_M is satisfiable, then we have an assignment for the elements of the trees of known shape, and by the sufficient surjectivity criterion we know that we can find a satisfying assignment for the other ones which will satisfy all disequalities of D.

We conclude by pointing out that the set abstraction, the multiset abstraction, the in-order traversal list abstraction and the sortedness abstraction are all infinitely surjective [SDK10].

5 Combining Theories with Shared Set Operations

We have seen several expressive decidable logics that are useful for specifying correctness properties of software and thus enable automated software verification. The correctness properties that are of practical interest often cannot be expressed in any single one of these logics, but only in their combination. This raises the question whether there exist decidable combinations of these logics and whether the decision procedure for such a combination can reuse the decision procedures for the component logics, e.g., in the style of the approach pioneered by Nelson and Oppen [NO79]. The Nelson-Oppen approach is one of the pillars of modern constraint solvers based on satisfiability modulo theories (SMT) [dMB08, BT07, GBT07]. It enables the combination of quantifier-free

stably infinite theories with disjoint signatures. However, the theories that we considered in the previous sections do not fit into this framework because they all involve sets of objects and are therefore not disjoint.

To support a broader class of theories than the traditional Nelson-Oppen combination, we consider decision procedures for the combination of *possibly quantified* formulas in *non-disjoint* theories. In [WPK09] we explored the case of the combination of non-disjoint theories sharing operations on *sets of uninterpreted elements*, a case that was not considered before. The theories that we consider have the property that the tuples of cardinalities of Venn regions over shared set variables in the models of a formula are a semilinear set (i.e., expressible in Presburger arithmetic).

Reduction-based decision procedure. The idea of deciding a combination of logics 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 $A \wedge B$.

We follow a reduction approach to decision procedures. The first decision procedure, D_A, computes a *projection*, S_A, of A onto *shared* set variables, which are free in both A and B. This projection is semantically equivalent to existentially quantifying over predicates and variables that are free in A but not in B; it is the strongest consequence of A expressible only using the shared set variables. D_B similarly computes the projection S_B of B. This reduces the satisfiability of $A \wedge B$ to satisfiability of the formula $S_A \wedge S_B$, which contains only set variables.

A logic for shared constraints on sets. A key parameter of our combination approach is the logic of sets used to express the projections S_A and S_B. A suitable logic depends on the logics of formulas A and B. We consider as the logics for A, B the logics we have discussed in the previous sections and other expressive logics we found useful based on our experience with the Jahob verification system [ZKR08, Wie09]. Remarkably, the smallest logic needed to express the projection formulas in these logics has the expressive power of BAPA, described in Section 2. We showed that the decision procedures for these logics can be naturally extended to a reduction to BAPA that captures precisely the constraints on set variables. The existence of these reductions, along with quantifier elimination [KNR06] and NP membership of the quantifier-free fragment [KR07], make BAPA an appealing reduction target for expressive logics.

We proved that 1) (quantified) Boolean Algebra with Presburger Arithmetic (Section 2), 2) quantifier-free multisets with cardinality constraints (Section 3), 3) weak monadic second-order logic of trees [TW68], 4) two-variable logic with counting C^2 [PH05], 5) the Bernays-Schönfinkel-Ramsey-class of first-order logic [Ram30], and 6) certain algebraic data types with abstraction functions (Section 4), all meet the conditions of our combination technique. Consequently, we obtain the decidability of quantifier-free combination of formulas in these logics. In the following we give an overview of our combination technique.

$$
\begin{array}{l}
\mathsf{tree}\,[\,\mathsf{left}\;,\mathsf{right}\,]\;\wedge\;\mathsf{left}\;\mathsf{p} = \mathsf{null}\;\wedge\;\mathsf{p} \in \mathsf{nodes}\;\wedge \\
\mathsf{nodes}{=}\{\mathsf{x}.\;(\mathsf{root},\mathsf{x})\;\in\{(\mathsf{x},\mathsf{y}).\;\mathsf{left}\;\;\mathsf{x} = \mathsf{y}|\mathsf{right}\;\;\mathsf{x} = \mathsf{y}\}\hat{}*\}\;\wedge \\
\mathsf{content}{=}\{\mathsf{x}.\;\exists\;\mathsf{n}.\;\mathsf{n} \neq \mathsf{null}\;\wedge\;\mathsf{n} \in \mathsf{nodes}\;\wedge\;\mathsf{data}\;\mathsf{n} = \mathsf{x}\}\;\wedge \\
\mathsf{e} \notin \mathsf{content}\;\wedge\;\mathsf{nodes} \subseteq \mathsf{alloc}\;\wedge \\
\mathsf{tmp} \notin \mathsf{alloc}\;\wedge\;\mathsf{left}\;\mathsf{tmp} = \mathsf{null}\;\wedge\;\mathsf{right}\;\mathsf{tmp} = \mathsf{null}\;\wedge \\
\mathsf{data}\;\mathsf{tmp} = \mathsf{null}\;\wedge\;(\forall\;\mathsf{y}.\;\mathsf{data}\;\mathsf{y} \neq \mathsf{tmp})\;\wedge \\
\mathsf{nodes1}{=}\{\mathsf{x}.\;(\mathsf{root},\mathsf{x})\;\in\{(\mathsf{x},\mathsf{y}).\;(\mathsf{left}\;\;(\mathsf{p}{:=}\mathsf{tmp}))\;\mathsf{x} = \mathsf{y})\;|\;\mathsf{right}\;\;\mathsf{x} = \mathsf{y}\}\;\wedge \\
\mathsf{content1}{=}\{\mathsf{x}.\;\exists\;\mathsf{n}.\;\mathsf{n} \neq \mathsf{null}\;\wedge\;\mathsf{n} \in \mathsf{nodes1}\;\wedge\;(\mathsf{data}(\mathsf{tmp}{:=}\mathsf{e}))\;\mathsf{n} = \mathsf{x}\}\;\rightarrow \\
\qquad\mathsf{card}\;\;\mathsf{content1} = \mathsf{card}\;\mathsf{content} + 1
\end{array}
$$

Fig. 9. Verification condition

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. 10. Negation of Fig. 9, and consequences on shared sets

5.1 Example: Proving a Verification Condition

Our example shows a verification condition formula generated when verifying an unbounded linked data structure. The formula belongs to our new decidable class obtained by combining several decidable logics.

Decidability of the verification condition. Fig. 9 shows the verification condition formula for a method (`insertAt`) that inserts a node into a linked list. The validity of this formula implies that invoking a method in a state satisfying the precondition results in a state that satisfies the postcondition 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. 9. 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. 9, consider the problem of showing the unsatisfiability of the negation of the formula. Fig. 10 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) 2) two-variable logic with counting C^2 3) Boolean Algebra with Presburger Arithmetic (BAPA). For the formula in each of the fragments, Fig. 10 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.)

A decision procedure. Note that the conjunction of the consequences of three formula fragments is an unsatisfiable formula. This shows that the original formula is unsatisfiable as well (the 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 "precise" consequences of formulas involving given shared sets. When a precise consequence is satisfiable in some assignment to set variables, it means that the original formula is also satisfiable with the same values of set variables. The consequences are all expressed in BAPA, which is decidable. In summary, the following is a decision procedure for satisfiability of combined formulas:

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

5.2 Combination by Reduction to BAPA

The Satisfiability Problem. We are interested in an algorithm to determine whether there exists a structure $\alpha \in \mathcal{M}$ in which the following formula is true

$$B(F_1, \ldots, F_n) \tag{1}$$

where

1. $F_1, \ldots, F_n$ are formulas with $\mathsf{FV}(F_i) \subseteq \{A_1, \ldots, A_p, x_1, \ldots, x_q\}$.
2. $V_S = \{A_1, \ldots, A_p\}$ are variables of sort set, whereas $x_1, \ldots, x_q$ are the remaining variables.[4]
3. Each formula F_i belongs to a given class of formulas, $\mathcal{F}_i$. For each $\mathcal{F}_i$ we assume that there is a corresponding theory $\mathcal{T}_i \subseteq \mathcal{F}_i$.
4. $B(F_1, \ldots, F_n)$ denotes a formula built from $F_1, \ldots, F_n$ using the propositional operations $\wedge, \vee$. [5]
5. As the set of structures $\mathcal{M}$ we consider all structures α of interest (with finite $[\![\mathsf{obj}]\!]$, interpreting BAPA symbols in the standard way) for which $\alpha(\cup_{i=1}^{n} \mathcal{T}_i)$.
6. (Set Sharing Condition) If $i \neq j$, then $\mathsf{FV}(\{F_i\} \cup \mathcal{T}_i) \cap \mathsf{FV}(\{F_j\} \cup \mathcal{T}_j) \subseteq V_S$.

Note that, as a special case, if we embed a class of first-order formulas into our framework, we obtain a framework that supports sharing unary predicates, but not e.g. binary predicates.

Combination Theorem. The formula B in (1) is satisfiable iff one of the disjuncts in its disjunctive normal form is satisfiable. Consider any of the disjuncts

[4] For notational simplicity we do not consider variables of sort obj because they can be represented as singleton sets, of sort set.

[5] The absence of negation is usually not a loss of generality because most $\mathcal{F}_i$ are closed under negation so B is the negation-normal form of a quantifier-free combination.

$F_1 \wedge \ldots \wedge F_m$ for $m \leq n$. By definition of the satisfiability problem (1), $F_1 \wedge \ldots \wedge F_m$ is satisfiable iff there exists a structure α such that for each $1 \leq i \leq m$, for each $G \in \{F_i\} \cup \mathcal{T}_i$, we have $\alpha(G) = \mathsf{true}$. Let each variable x_i have some sort s_i (such as $\mathsf{obj}^2 \to \mathsf{bool}$). Then the satisfiability of $F_1 \wedge \ldots \wedge F_m$ is equivalent to the following condition:

$$\begin{array}{l} \exists \text{ finite set } u.\ \exists a_1, \ldots, a_p \subseteq u.\ \exists v_1 \in [\![s_1]\!]^u. \ldots.\ \exists v_q \in [\![s_q]\!]^u.\ \bigwedge_{i=1}^m \\ \{\mathsf{obj} \to u, A_1 \mapsto a_1, \ldots, A_p \mapsto a_p, x_1 \mapsto v_1, \ldots, x_q \mapsto v_q\}(\{F_i\} \cup \mathcal{T}_i) \end{array} \quad (2)$$

By the set sharing condition, each of the variables $x_1, \ldots, x_q$ appears only in one conjunct and can be moved inwards from the top level to this conjunct. Using x_{ij} to denote the j-th variable in the i-th conjunct we obtain the condition

$$\exists \text{ finite set } u.\ \exists a_1, \ldots, a_p \subseteq u.\ \bigwedge_{i=1}^m C_i(u, a_1, \ldots, a_p) \quad (3)$$

where $C_i(u, a_1, \ldots, a_p)$ is

$$\begin{array}{l} \exists v_{i1} \ldots \exists v_{iw_i}. \\ \{\mathsf{obj} \to u, A_1 \mapsto a_1, \ldots, A_p \mapsto a_p, x_{i1} \mapsto v_{i1}, \ldots, x_{iw_i} \mapsto v_{iw_i}\}(\{F_i\} \cup \mathcal{T}_i) \end{array}$$

The idea of our combination method is to simplify each condition $C_i(u, a_1, \ldots, a_p)$ into the truth value of a BAPA formula. If this is possible, we say that there exists a BAPA reduction.

Definition 3 (BAPA Reduction). *If $\mathcal{F}_i$ is a set of formulas and $\mathcal{T}_i \subseteq \mathcal{F}_i$ a theory, we call a function $\rho : \mathcal{F}_i \to \mathcal{F}_{\mathsf{BAPA}}$ a BAPA reduction for $(\mathcal{F}_i, \mathcal{T}_i)$ iff for every formula $F_i \in \mathcal{F}_i$ and for all finite u and $a_1, \ldots, a_p \subseteq u$, the condition*

$$\begin{array}{l} \exists v_{i1} \ldots \exists v_{iw_i}. \\ \{\mathsf{obj} \to u, A_1 \mapsto a_1, \ldots, A_p \mapsto a_p, x_{i1} \mapsto v_{i1}, \ldots, x_{iw_i} \mapsto v_{iw_i}\}(\{F_i\} \cup \mathcal{T}_i) \end{array}$$

is equivalent to the condition $\{\mathsf{obj} \to u, A_1 \mapsto a_1, \ldots, A_p \mapsto a_p\}(\rho(F_i))$.

A computable BAPA reduction is a BAPA reduction which is computable as a function on formula syntax trees.

Theorem 4. *Suppose that for every $1 \leq i \leq n$ for $(\mathcal{F}_i, \mathcal{T}_i)$ there exists a computable BAPA reduction ρ_i. Then the satisfiability problem* (1) *is decidable.*

Specifically, to check satisfiability of the formula $B(F_1, \ldots, F_n)$, compute $B(\rho_1(F_1), \ldots, \rho_n(F_n))$ and then check its satisfiability using a BAPA decision procedure [KNR06, KR07].

5.3 BAPA Reductions

The proof that a particular decidable logic exhibits a BAPA reduction follows a generic recipe. Given such a logic $\mathcal{L} = (\mathcal{F}, \mathcal{T})$ and a formula $F \in \mathcal{F}$, let $V_1, \ldots, V_n$ be the Venn regions over the free set variables in F. To prove that $\mathcal{L}$ is BAPA-reducible, one needs to characterize the cardinality vectors of the V_i in all the

models of F: $\mathcal{V}(F) = \{ (| \alpha(V_1) |, \ldots, | \alpha(V_n) |) \mid \alpha(\mathcal{T} \cup \{F\}) = 1 \}$ and show that this set is semilinear. Moreover, a finite representation of the set $\mathcal{V}(F)$ in terms of base and set vectors must be effectively computable from F, by extending the decision procedure for $\mathcal{L}$ appropriately. We have shown [WPK09, Theorems 5, 11, 12, 13], [SDK10] that the decision procedures for a number of expressive decidable logics can indeed be extended in this way to BAPA reductions.

Theorem 5. *There exist BAPA reductions for the following logics (see Figure 1)*

1. *weak monadic second-order logic of trees [TW68]*
2. *two-variable logic with counting C^2 [PH05]*
3. *the Bernays-Schönfinkel-Ramsey class of first-order logic [Ram30]*
4. *quantifier-free multisets with cardinality constraints (Figure 4)*
5. *logic of algebraic data types with the* content *function (in Figure 7)*

Thus, the set-sharing combination of all these logics is decidable.

References

[BGG97] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.

[BM07] Aaron R. Bradley and Zohar Manna. *The Calculus of Computation*. Springer, 2007.

[BST07] Clark Barrett, Igor Shikanian, and Cesare Tinelli. An abstract decision procedure for satisfiability in the theory of recursive data types. *Electronic Notes in Theoretical Computer Science*, 174(8):23–37, 2007.

[BT07] Clark Barrett and Cesare Tinelli. CVC3. In *CAV*, volume 4590 of *LNCS*, 2007.

[CD94] Hubert Comon and Catherine Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, 1994.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.

[dMB09] Leonardo de Moura and Nikolaj Bjørner. Generalized, efficient array decision procedures. In *FMCAD*, 2009.

[ES06] Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Operations Research Letters*, 34(5):564–568, September 2006.

[GBT07] Yeting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In *CADE*, 2007.

[GHN+04] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In *CAV*, pages 175–188, 2004.

[GS66] S. Ginsburg and E. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

[KNR06] Viktor Kuncak, Hai Huu Nguyen, and Martin Rinard. Deciding Boolean Algebra with Presburger Arithmetic. *J. of Automated Reasoning*, 2006.

[KR07] Viktor Kuncak and Martin Rinard. Towards efficient satisfiability checking for Boolean Algebra with Presburger Arithmetic. In *CADE-21*, 2007.

[LS04] Shuvendu K. Lahiri and Sanjit A. Seshia. The UCLID decision procedure. In *CAV'04*, 2004.

[NO79] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM TOPLAS*, 1(2):245–257, 1979.

[Opp78] Derek C. Oppen. Reasoning about recursively defined data structures. In *POPL*, pages 151–157, 1978.

[OSV08] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: a comprehensive step-by-step guide*. Artima Press, 2008.

[Pap81] Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

[PH05] Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.

[PK08a] Ruzica Piskac and Viktor Kuncak. Decision procedures for multisets with cardinality constraints. In *VMCAI*, 2008.

[PK08b] Ruzica Piskac and Viktor Kuncak. Fractional collections with cardinality bounds. In *Computer Science Logic (CSL)*, 2008.

[PK08c] Ruzica Piskac and Viktor Kuncak. Linear arithmetic with stars. In *CAV*, 2008.

[Pot91] Loïc Pottier. Minimal solutions of linear diophantine systems: Bounds and algorithms. In *RTA*, volume 488 of *LNCS*, 1991.

[Pre29] Mojzesz Presburger. Über die Vollständigkeit eines gewissen Systems der Aritmethik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du premier Congrès des Mathématiciens des Pays slaves, Warsawa*, pages 92–101, 1929.

[PST00] Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity results for first-order two-variable logic with counting. *SIAM J. on Computing*, 29(4):1083–1117, 2000.

[Ram30] F. P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc.*, s2-30:264–286, 1930. doi:10.1112/plms/s2-30.1.264.

[SBDL01] Aaron Stump, Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. A decision procedure for an extensional theory of arrays. In *LICS*, pages 29–37, 2001.

[SDK10] Philippe Suter, Mirco Dotta, and Viktor Kuncak. Decision procedures for algebraic data types with abstractions. In *POPL*, 2010.

[Sko19] Thoralf Skolem. Untersuchungen über die Axiome des Klassenkalküls and über "Produktations- und Summationsprobleme", welche gewisse Klassen von Aussagen betreffen. Skrifter utgit av Vidnskapsselskapet i Kristiania, I. klasse, no. 3, Oslo, 1919.

[TW68] 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, August 1968.

[Wie09] Thomas Wies. *Symbolic Shape Analysis*. PhD thesis, University of Freiburg, 2009.

[WPK09] Thomas Wies, Ruzica Piskac, and Viktor Kuncak. Combining theories with shared set operations. In *FroCoS: Frontiers in Combining Systems*, 2009.

[ZKR08] Karen Zee, Viktor Kuncak, and Martin Rinard. Full functional verification of linked data structures. In *PLDI*, 2008.