Deciding Boolean Algebra with Presburger Arithmetic

Viktor Kuncak[1], Huu Hai Nguyen[2], and Martin Rinard [1,2]
[1] *MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, USA*
[2] *Singapore-MIT Alliance*

Abstract. We describe an algorithm for deciding the first-order multisorted theory BAPA, which combines 1) Boolean algebras of sets of uninterpreted elements (BA) and 2) Presburger arithmetic operations (PA). BAPA can express the relationship between integer variables and cardinalities of unbounded finite sets, and supports arbitrary quantification over sets and integers.

Our original motivation for BAPA is deciding verification conditions that arise in the static analysis of data structure consistency properties. Data structures often use an integer variable to keep track of the number of elements they store; an invariant of such a data structure is that the value of the integer variable is equal to the number of elements stored in the data structure. When the data structure content is represented by a set, the resulting constraints can be captured in BAPA. BAPA formulas with quantifier alternations arise when verifying programs with annotations containing quantifiers, or when proving simulation relation conditions for refinement and equivalence of program fragments. Furthermore, BAPA constraints can be used for proving the termination of programs that manipulate data structures, as well as in constraint database query evaluation and loop invariant inference.

We give a formal description of an algorithm for deciding BAPA. We analyze our algorithm and show that it has optimal alternating time complexity, and that the complexity of BAPA matches the complexity of PA. Because it works by a reduction to PA, our algorithm yields the decidability of a combination of sets of uninterpreted elements with any decidable extension of PA. When restricted to BA formulas, the algorithm can be used to decide BA in optimal alternating time. Furthermore, the algorithm can eliminate individual quantifiers from a formula with free variables and therefore perform projection onto a desirable set of variables.

We have implemented our algorithm and used it to discharge verification conditions in the Jahob system for data structure consistency checking of Java programs; our experience suggest that a straightforward implementation of the algorithm is effective on non-trivial formulas as long as the number of set variables is small. We also report on a new algorithm for solving the quantifier-free fragment of BAPA.

Keywords: Boolean algebra, Presburger arithmetic, decision procedure, quantifier elimination, complexity, program verification

1. Introduction

Program analysis and verification tools can greatly contribute to software reliability, especially when used throughout the software development process. Such tools are even more valuable if their behavior is predictable, if they can be applied to partial programs, and if they allow the developer to communicate the design information in the form of specifications. Combining the basic idea of (Hoare, 1969) with decidable logics leads to analysis tools that have these desirable properties. Such analyses are precise (because formulas represent loop-free code precisely) and predictable (because the checking of verification conditions terminates either with a counterexample or with a proof that there are no counterexamples).

A key challenge in this approach to program analysis and verification is to identify a logic that captures an interesting class of program properties, but is nevertheless decidable. In (Lam et al., 2005) we identify the first-order theory of Boolean algebras (BA) as a useful language for reasoning about dynamically allocated objects: BA allows expressing generalized typestate properties and reasoning about data structures as dynamically changing sets of objects. Here we are interested in BA of all subsets of some set; this theory was shown decidable already in (Loewenheim, 1915; Skolem, 1919), see (Kozen, 1980) for the decidability and the complexity of all models of BA axioms.

The motivation for this paper is the fact that we often need to reason not only about the content of a data structure, but also about the size of a data structure. For example, we may want to express the fact that the number of elements stored in a data structure is equal to the value of an integer variable that is used to cache the data structure size, or we may want to introduce a decreasing integer measure on the data structure to show program termination. These considerations lead to a natural generalization of the first-order theory of BA of sets, a generalization that allows integer variables in addition to set variables, and allows stating relations of the form $|A| = k$ meaning that the cardinality of the set A is equal to the value of the integer variable k. Once we have integer variables, a natural question arises: which relations and operations on integers should we allow? It turns out that, using only the BA operations and the cardinality operator, we can already define all operations of PA. This leads to the structure BAPA, which properly generalizes both BA and PA.

As we explain in Section 2, a version of BAPA was shown decidable already in (Feferman and Vaught, 1959). Recently, a decision procedure for a fragment of BAPA without quantification over sets was presented in (Zarba, 2004c), cast as a multi-sorted theory. Starting from (Lam et al., 2005) as our motivation, we used quantifier elimination in (Kuncak and Rinard, 2004) to show the decidability of the full BAPA, which was initially stated as an open question in (Zarba, 2004c). A quantifier-elimination algorithm for a single-sorted version of BAPA was presented independently in (Revesz, 2004) as a way of evaluating queries in constraint databases; (Revesz, 2004) leaves open the complexity of the decision problem.

Our paper gives the first formal description of a decision procedure for the full first-order theory of BAPA. Furthermore, we analyze our decision procedure and show that it yields optimal computational complexity for BAPA, identical to the complexity of PA (Berman, 1980). This solves the question of the computational complexity of BAPA. [1] We have also implemented our decision procedure; we report on our initial experience in using the decision procedure in a system for automatic verification of imperative data structures.

1.1. Contributions

We summarize the contributions of our paper as follows.

1. As a **motivation** for BAPA, we show in Section 3 that BAPA constraints can be used for program analysis and verification by expressing 1) data structure invariants and the correctness of procedures with respect to their specifications,[2] 3) simulation relations between program fragments, 4) termination conditions for programs that manipulate data structures, and 5) projection of formulas onto a desired set of variables, with applications in static analysis, model checking, automated abstraction, and relational query evaluation.

2. We present an **algorithm** α (Section 4) that translates BAPA sentences into PA sentences by translating set quantifiers into integer quantifiers and show how it can be used to decide the truth value of PA formulas and to eliminate individual quantifiers (Section 6).

3. We analyze our algorithm α and show that its **complexity** matches the lower bound for PA and is therefore **optimal** (Section 5). This result solves the question of the complexity of the decision problem for BAPA and is the main technical contribution of this paper. Our analysis includes showing an alternating time upper bound for PA, parameterized by the number of quantifier alternations.

4. We discuss our initial experience in using our **implementation** of BAPA to discharge verification conditions generated in the Jahob verification system (Kuncak and Rinard, 2006).

5. We observe the following additional results:

[1] In (Kuncak et al., 2005) we have obtained only the corresponding space upper bound on BAPA; we thank Dexter Kozen for suggesting to use an alternating complexity class of (Berman, 1980) to establish the optimal bound.

[2] This motivation was presented first in (Kuncak and Rinard, 2004) and was subsequently used in (Zarba, 2005).

a) PA sentences generated by translating BA sentences without cardinalities can be decided in optimal alternating time for BA (Section 5.4), which gives an alternative proof of upper **bound for** BA **of sets**;

b) Our algorithm extends to **countable sets** with a predicate distinguishing finite and infinite sets (Section 8);

c) In contrast to the undecidability of monadic second-order logic over strings (MSOL) when extended with equicardinality operator, we identify a decidable combination of **MSOL with** BA (Section 8).

d) We outline recent results on solving quantifier-free fragment of BAPA using a technique that, for the first time, avoids explicitly constructing an exponentially large system of integer constraints. Our technique yields a polynomial space upper bound on **quantifier-free fragment of** BAPA.

A preliminary version of our results, including the algorithm and complexity analysis appear in (Kuncak and Rinard, 2004; Kuncak et al., 2005) along with additional details on quantifier elimination. The recent results on quantifier-free fragment of BAPA are described in (Marnette et al., 2005, Section 3).

2. The First-Order Theory BAPA

Figure 3 presents the syntax of Boolean Algebra with Presburger Arithmetic (BAPA), which is the focus of this paper. We next present some justification for the operations in Figure 3. Our initial motivation for BAPA was the use of BA to reason about data structures in terms of sets (Lam et al., 2004). Our language for BA (Figure 1) allows cardinality constraints of the form $|A| = K$ where K is a *constant* integer. Such constant cardinality constraints are useful and enable quantifier elimination for the resulting language (Skolem, 1919; Loewenheim, 1915). However, they do not allow stating constraints such as $|A| = |B|$ for two sets A and B, and cannot represent constraints on relationships between sizes of sets and integer variables. Consider therefore the equicardinality relation $A \sim B$ that holds iff $|A| = |B|$, and consider BA extended with relation $A \sim B$. Define the ternary relation $\mathsf{plus}(A, B, C) \iff (|A|+|B| = |C|)$ by the formula $\exists x_1. \exists x_2. x_1 \cap x_2 = \emptyset \ \wedge \ A \sim x_1 \ \wedge \ B \sim x_2 \ \wedge \ x_1 \cup x_2 = C$. The relation $\mathsf{plus}(A, B, C)$ allows us to express addition using arbitrary sets as representatives for natural numbers; $\emptyset$ can represent the natural number zero, and any singleton set can represent the natural number one. (The property

$$
\begin{aligned}
F &::= A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg F \mid \exists x.F \mid \forall x.F \\
A &::= B_1 = B_2 \mid B_1 \subseteq B_2 \mid |B| = K \ \mid \ |B| \geq K \\
B &::= x \mid \mathbf{0} \mid \mathbf{1} \mid B_1 \cup B_2 \mid B_1 \cap B_2 \mid B^c \\
K &::= 0 \mid 1 \mid 2 \mid \ldots
\end{aligned}
$$

Figure 1. Formulas of Boolean Algebra (BA)

$$
\begin{aligned}
F &::= A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg F \mid \exists k.F \mid \forall k.F \\
A &::= T_1 = T_2 \mid T_1 < T_2 \mid K \,\mathsf{dvd}\, T \\
T &::= K \mid T_1 + T_2 \mid K \cdot T \\
K &::= \ldots -2 \mid -1 \mid 0 \mid 1 \mid 2 \ldots
\end{aligned}
$$

Figure 2. Formulas of Presburger Arithmetic (PA)

of A being a singleton is definable using e.g. the first-order formula $A \neq \emptyset \wedge \forall B.A \cap B = B \Rightarrow (B = \emptyset \vee B = A)$.) Moreover, we can represent integers as equivalence classes of pairs of natural numbers under the equivalence relation $(x, y) \approx (u, v) \iff x + v = u + y$; this construction also allows us to express the unary predicate of being non-negative. The quantification over pairs of sets represents quantification over integers, and quantification over integers with the addition operation and the predicate "being non-negative" can express all PA operations, presented in Figure 2. Therefore, a natural closure under definable operations leads to our formulation of the language BAPA in Figure 3, which contains both sets and integers.

The argument above also explains why we attribute the decidability of BAPA to (Feferman and Vaught, 1959, Section 8), which showed the decidability of BA over sets extended with the equicardinality relation $\sim$, using the decidability of the first-order theory of the addition of cardinal numbers.

$$
\begin{aligned}
F &::= A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg F \mid \exists x.F \mid \forall x.F \mid \exists k.F \mid \forall k.F \\
A &::= B_1 = B_2 \mid B_1 \subseteq B_2 \mid T_1 = T_2 \mid T_1 < T_2 \mid K \,\mathsf{dvd}\, T \\
B &::= x \mid \mathbf{0} \mid \mathbf{1} \mid B_1 \cup B_2 \mid B_1 \cap B_2 \mid B^c \\
T &::= k \mid K \mid \mathsf{MAXC} \mid T_1 + T_2 \mid K \cdot T \mid \ |B| \\
K &::= \ldots -2 \mid -1 \mid 0 \mid 1 \mid 2 \ldots
\end{aligned}
$$

Figure 3. Formulas of Boolean Algebra with Presburger Arithmetic (BAPA)

The language BAPA has two kinds of quantifiers: quantifiers over integers and quantifiers over sets; we distinguish between these two kinds by denoting integer variables with symbols such as k, l and set variables with symbols such as x, y. We use the shorthand $\exists^+ k.F(k)$ to denote $\exists k.k \geq 0 \wedge F(k)$ and, similarly $\forall^+ k.F(k)$ to denote $\forall k.k \geq 0 \Rightarrow F(k)$. In summary, the language of BAPA in Figure 3 subsumes the language of PA in Figure 2, subsumes the language of BA in Figure 1, and contains non-trivial combination of these two languages in the form of using the cardinality of a set expression as an integer value.

The semantics of operations in Figure 3 is the expected one. We interpret integer terms as integers, and interpret set terms as elements of the powerset of a finite set. The MAXC constant denotes the size of the finite universe $\mathcal{U}$, so we require $\mathsf{MAXC} = |\mathcal{U}|$ in all models. Our results generalize to the Boolean algebra of powersets of a countable set, see Section 8.

3. Applications of BAPA

This section illustrates the importance of BAPA constraints. Section 3.1 shows the uses of BAPA constraints to express and verify data structure invariants as well as procedure preconditions and postconditions. Section 3.2 shows how a class of simulation relation conditions can be proved automatically using a decision procedure for BAPA. Section 3.3 shows how BAPA can be used to express and prove termination conditions for a class of programs. Section 3.4 discusses the applications of quantifier elimination, which is relevant to BAPA because our decision procedure is based on quantifier elimination.

3.1. Verifying Data Structure Consistency

Figure 4 presents a procedure insert in a language that directly manipulates sets. Such languages can either be directly executed (Dewar, 1979) or can arise as abstractions of programs in standard languages (Lam et al., 2005). The program in Figure 4 manipulates a global set of objects content and an integer field size. The program maintains an invariant I that the size of the set content is equal to the value of the variable size. The insert procedure inserts an element e into the set and correspondingly updates the integer variable. The requires clause (precondition) of the insert procedure is that the parameter e is a non-null reference to an object that is not stored in the set content. The ensures clause (postcondition) of the procedure is that the size variable after the insertion is positive. Note that we represent references to objects (such

$$
\begin{array}{l}
\textbf{var } \mathsf{content} : \textbf{set}; \\
\textbf{var } \mathsf{size} : \textbf{integer}; \\
\textbf{invariant } I \iff (\mathsf{size} = |\mathsf{content}|); \\
\\
\textbf{procedure } \mathsf{insert}(e : \mathsf{element}) \\
\textbf{maintains } I \\
\textbf{requires } |e| = 1 \land |e \cap \mathsf{content}| = 0 \\
\textbf{ensures } \mathsf{size}' > 0 \\
\{ \\
\quad \mathsf{content} := \mathsf{content} \cup e; \\
\quad \mathsf{size} := \mathsf{size} + 1; \\
\}
\end{array}
$$

Figure 4. An Example Procedure

$$
\begin{array}{l}
\Big\{ |e| = 1 \land |e \cap \mathsf{content}| = 0 \land \mathsf{size} = |\mathsf{content}| \Big\} \\
\quad \mathsf{content} := \mathsf{content} \cup e; \quad \mathsf{size} := \mathsf{size} + 1; \\
\Big\{ \mathsf{size}' > 0 \land \mathsf{size}' = |\mathsf{content}'| \Big\}
\end{array}
$$

Figure 5. Hoare Triple for insert Procedure

as the procedure parameter e) as sets with at most one element. An empty set represents a null reference; a singleton set $\{o\}$ represents a reference to object o. The value of a variable after procedure execution is indicated by marking the variable name with a prime.

The insert procedure maintains an invariant, I, which captures the relationship between the size of the set content and the integer variable size. The invariant I is implicitly conjoined with the requires and the ensures clauses of the procedure. The Hoare triple in Figure 5 summarizes the resulting correctness condition for the insert procedure. Figure 6 presents a verification condition corresponding to the Hoare triple in Figure 5. Note that the verification condition contains both set and integer variables, contains quantification over these variables, and relates the sizes of sets to the values of integer variables. Our small example leads to a formula without quantifier alternations; in general, formulas that arise in verification may contain alternations of existential and universal variables over both integers and sets. This paper

$$
\begin{array}{l}
\forall e.\ \forall \mathsf{content}.\ \forall \mathsf{content}'.\ \forall \mathsf{size}.\ \forall \mathsf{size}'. \\
\quad (|e| = 1 \land |e \cap \mathsf{content}| = 0 \land \mathsf{size} = |\mathsf{content}| \land \\
\quad\ \mathsf{content}' = \mathsf{content} \cup e \land \mathsf{size}' = \mathsf{size} + 1) \Rightarrow \\
\qquad \mathsf{size}' > 0 \land \mathsf{size}' = |\mathsf{content}'|
\end{array}
$$

Figure 6. Verification Condition for Figure 5

shows the decidability of such formulas and presents the complexity of the decision procedure.

3.2. Proving Simulation Relation Conditions

BAPA constraints are also useful when proving that a given binary relation on states is a simulation relation between two program fragments. Figure 7 shows one such example. The concrete procedure start1 manipulates two sets: a set of running processes and a set of suspended processes in a process scheduler. The procedure start1 inserts a new process x into the set of running processes R, unless there are already too many running processes. The procedure start2 is a version of the procedure that operates in a more abstract state space: it maintains only the union P of all processes and the number k of running processes. Figure 7 shows a forward simulation relation r between the transition relations for start1 and start2. The standard simulation relation diagram condition is $\forall s_1.\forall s_1'.\forall s_2.(t_1(s_1,s_1') \land r(s_1,s_2)) \Rightarrow \exists s_2'.\ (t_2(s_2,s_2') \land r(s_1',s_2'))$. In the presence of preconditions, $t_1(s_1,s_1') = (\mathsf{pre}_1(s_1) \Rightarrow \mathsf{post}_1(s_1,s_1'))$ and $t_2(s_2,s_2') = (\mathsf{pre}_2(s_2) \Rightarrow \mathsf{post}_2(s_2,s_2'))$, and sufficient conditions for simulation relation are:

1. $\forall s_1.\forall s_2.r(s_1,s_2) \land \mathsf{pre}_2(s_2) \Rightarrow \mathsf{pre}_1(s_1)$
2. $\forall s_1.\forall s_1'.\forall s_2.\exists s_2'.\ r(s_1,s_2) \land \mathsf{post}_1(s_1,s_1') \land \mathsf{pre}_2(s_2)$
   $\Rightarrow \mathsf{post}_2(s_2,s_2') \land r(s_1',s_2')$

Figure 7 shows BAPA formulas that correspond to the simulation relation conditions in this example. Note that the second BAPA formula has a quantifier alternation, which illustrates the relevance of quantifiers in BAPA.

3.3. Proving Program Termination

We next show that BAPA is useful for proving program termination. A standard technique for proving termination of a loop is to introduce a ranking function f that maps program states to non-negative integers, then prove that the value of the function decreases at each loop iteration. In other words, if $t(s,s')$ denotes the relationship between the state at the beginning and the state at the end of each loop iteration, then the condition $\forall s.\forall s'.t(s,s') \Rightarrow f(s) > f(s')$ holds. Figure 8 shows an example program that processes each element of the initial value of set iter; this program can be viewed as manipulating an iterator over a data structure that implements a set. Using the the ability to take cardinality of a set allows us to define a natural ranking function for this program. Figure 9 shows the termination proof based on such ranking function. The resulting termination condition can be expressed as a formula that

```
var P : set;
var k : integer;

procedure start2(x)
requires x ⊈ P ∧ |x| = 1 ∧ k < MAXR
ensures P' = P ∪ x ∧ k' = k + 1
{
  P := P ∪ x;
  k := k + 1;
}
```

```
var R : set;
var S : set;

procedure start1(x)
requires x ⊈ R ∧ |x| = 1 ∧ |R| < MAXR
ensures R' = R ∪ x ∧ S' = S
{
  R := R ∪ x;
}
```

Simulation relation *r:*
$r((\mathsf{R}, \mathsf{S}), (\mathsf{P}, \mathsf{k})) = (\mathsf{P} = \mathsf{R} \cup \mathsf{S} \wedge \mathsf{k} = |\mathsf{R}|)$

Simulation relation conditions in BAPA:

1. $\forall x, \mathsf{R}, \mathsf{S}, \mathsf{P}, \mathsf{k}.(\mathsf{P} = \mathsf{R} \cup \mathsf{S} \wedge \mathsf{k} = |\mathsf{R}|) \wedge (x \not\subseteq \mathsf{P} \wedge |x| = 1 \wedge \mathsf{k} < \mathsf{MAXR}) \Rightarrow (x \not\subseteq \mathsf{R} \wedge |x| = 1 \wedge |\mathsf{R}| < \mathsf{MAXR})$
2. $\forall x, \mathsf{R}, \mathsf{S}, \mathsf{R}', \mathsf{S}', \mathsf{P}, \mathsf{k}.\exists \mathsf{P}', \mathsf{k}'.((\mathsf{P} = \mathsf{R} \cup \mathsf{S} \wedge \mathsf{k} = |\mathsf{R}|) \wedge (\mathsf{R}' = \mathsf{R} \cup x \wedge \mathsf{S}' = \mathsf{S}) \wedge (x \not\subseteq \mathsf{P} \wedge |x| = 1 \wedge \mathsf{k} < \mathsf{MAXR})) \Rightarrow (\mathsf{P}' = \mathsf{P} \cup x \wedge \mathsf{k}' = \mathsf{k} + 1) \wedge (\mathsf{P}' = \mathsf{R}' \cup \mathsf{S}' \wedge \mathsf{k}' = |\mathsf{R}'|)$

Figure 7. Proving simulation relation in BAPA

```
var iter : set;

procedure iterate()
{
  while iter ≠ ∅ do
    var e : set;
    e := choose iter;
    iter := iter \ e;
    process(e);
  done
}
```

Figure 8. Terminating program

belongs to BAPA, and can be discharged using our decision procedure. In general, we can reduce the termination problem of programs that manipulate both sets and integers to showing a simulation relation with a fragment of a terminating program that manipulates only integers, which can be proved terminating using techniques (Podelski and Rybalchenko, 2005). The simulation relation condition can be proved correct using our BAPA decision procedure whenever the simulation relation is expressible with a BAPA formula. While one could, in principle, use finite sets directly to describe certain ranking functions, the

Ranking function:
$f(s) = |s|$

Transition relation:
$t(\mathsf{iter}, \mathsf{iter}') = (\exists e.\ |e| = 1 \land e \subseteq \mathsf{iter} \land \mathsf{iter}' = \mathsf{iter} \setminus e)$

Termination condition in BAPA:
$$\begin{array}{l}\forall \mathsf{iter}. \forall \mathsf{iter}'.\ (\exists e. |e| = 1 \land e \subseteq \mathsf{iter} \land \mathsf{iter}' = \mathsf{iter} \setminus e) \\ \qquad\qquad \Rightarrow |\mathsf{iter}'| < |\mathsf{iter}|\end{array}$$

Figure 9. Termination proof for Figure 8

$$\exists e.\ |e| = 1 \land \mathsf{content}' = \mathsf{content} \cup e$$
$$\Downarrow$$
$$\mathsf{content} \subseteq \mathsf{content}' \ \land\ |\mathsf{content}' \setminus \mathsf{content}| \leq 1 \ \land\ |\mathsf{content}'| \geq 1$$

Figure 10. Eliminating a local variable from a transition relation

ability to abstract sets into integers allows us to use existing tools and techniques developed for integer ranking functions.

3.4. Quantifier Elimination

The fact that BAPA admits quantifier elimination enables applications that involve hiding certain variables from a BAPA formula. Hiding a variable x in a formula means existentially quantifying over x and then eliminating the quantifier $\exists x$. This process can also be viewed as a projection of a formula onto variables other than x. As an example, Figure 10 shows the transition relation inspired by the procedure insert in Figure 5. The transition relation mentions a variable e that is local to the procedure and not meaningful outside it. In the public description of the transition relation the variable e is existentially quantified. Our quantifier elimination algorithm (Section 4, Section 6) removes the quantifier from the formula and obtains an equivalent formula without quantifiers, such as the one shown in the lower part of Figure 10.

In general, variable hiding is useful in projecting state transitions and invariants onto a desired set of variables, computing relational composition of transition relations, and computing the image of a set under a transition relation. Such symbolic computation of transition relations, with appropriate widenings, can be used to generalize static analyses such as (Lam et al., 2005) and model checking approaches such as (Bultan et al., 1999). The quantifier elimination process here ensures that the transition relation remains represented by a quantifier-free formula throughout the analysis.

Quantifier elimination is also useful for query evaluation in constraint databases (Revesz, 2004), and loop invariant inference (Kapur, 2004).

4. Decision Procedure for BAPA

This section presents our algorithm, denoted α, which decides the validity of BAPA sentences. The algorithm reduces a BAPA sentence to an equivalent PA sentence with the same number of quantifier alternations and an exponential increase in the total size of the formula. This algorithm has several desirable properties:

1. Given an optimal algorithm for deciding PA sentences, the algorithm α is optimal for deciding BAPA sentences and shows that the complexity of BAPA is the same as the complexity of PA (Section 5).

2. The algorithm α does not eliminate integer variables, but instead produces an equivalent quantified PA sentence. The resulting PA sentence can therefore be decided using *any* decision procedure for PA, including the decision procedures based on automata (Klarlund et al., 2000; Boigelot et al., 2005).

3. The algorithm α can eliminate set quantifiers from any extension of PA. We thus obtain a technique for adding a particular form of set reasoning to every extension of PA, and the technique preserves the decidability of the extension. One example of decidable theory that extends PA is MSOL over strings, see Section 8.

4. For simplicity we present the algorithm α as a decision procedure for formulas with no free variables, but the algorithm can be used to transform and simplify formulas with free variables as well, because it transforms one quantifier at a time starting from the innermost one. We explore this version of our algorithm in Section 6.

We next describe the algorithm α for transforming a BAPA sentence F_0 into a PA sentence. As the first step of the algorithm, transform F_0 into prenex form

$$Q_p v_p. \ldots Q_1 v_1.\ F(v_1, \ldots, v_p)$$

where F is quantifier-free, and each quantifier $Q_i v_i$ is of one the forms $\exists k$, $\forall k$, $\exists y$, $\forall y$ where k denotes an integer variable and y denotes a set variable.

The next step of the algorithm is to separate F into a BA part and a PA part. To achieve this, replace each formula $b_1 = b_2$ where b_1 and b_2 are set expressions, with the conjunction $b_1 \subseteq b_2 \wedge b_2 \subseteq b_1$, and replace each formula $b_1 \subseteq b_2$ with the equivalent formula $|b_1 \cap b_2^c| = 0$. In the resulting formula, each set variable x occurs in some term $|t(x)|$. Next, use the same reasoning as when generating disjunctive normal form for propositional logic to write each set expression $t(x)$ as a union of cubes (regions in the Venn diagram). The cubes have the form $\bigcap_{i=1}^n x_i^{\alpha_i}$ where $x_i^{\alpha_i}$ is either x_i or x_i^c; there are $m = 2^n$ cubes $s_1, \ldots, s_m$. Suppose that $t(x) = s_{j_1} \cup \ldots \cup s_{j_a}$; then replace the term $|t(x)|$ with the term $\sum_{i=1}^a |s_{j_i}|$. In the resulting formula, each set x appears in an expression of the form $|s_i|$ where s_i is a cube. For each s_i introduce a new variable l_i. The resulting formula is then equivalent to

$$\begin{array}{l} Q_p v_p. \ldots Q_1 v_1. \\ \qquad \exists^+ l_1, \ldots, l_m.\ \bigwedge_{i=1}^m |s_i| = l_i \ \wedge\ G_1 \end{array} \tag{1}$$

where G_1 is a PA formula. Formula (1) is the starting point of the main phase of algorithm α. The main phase of the algorithm successively eliminates quantifiers $Q_1 v_1, \ldots, Q_p v_p$ while maintaining a formula of the form

$$\begin{array}{l} Q_p v_p \ldots Q_r v_r. \\ \qquad \exists^+ l_1 \ldots l_q.\ \bigwedge_{i=1}^q |s_i| = l_i \ \wedge\ G_r \end{array} \tag{2}$$

where G_r is a PA formula, r grows from 1 to $p+1$, and $q = 2^e$ where e for $0 \leq e \leq n$ is the number of set variables among $v_p, \ldots, v_r$. The list $s_1, \ldots, s_q$ is the list of all 2^e partitions formed from the set variables among $v_p, \ldots, v_r$.

We next show how to eliminate the innermost quantifier $Q_r v_r$ from the formula (2). During this process, the algorithm replaces the formula G_r with a formula G_{r+1} which has more integer quantifiers. If v_r is an integer variable then the number of sets q remains the same, and if v_r is a set variable, then q reduces from 2^e to 2^{e-1}. We next consider each of the four possibilities $\exists k$, $\forall k$, $\exists y$, $\forall y$ for the quantifier $Q_r v_r$.

Case $\exists$k: Consider first the case $\exists k$. Because k does not occur in $\bigwedge_{i=1}^q |s_i| = l_i$, simply move the existential quantifier to G_r and let $G_{r+1} = \exists k. G_r$, which completes the step.

Case $\forall$k: For universal quantifiers, it suffices to let $G_{r+1} = \forall k. G_r$, again because k does not occur in $\bigwedge_{i=1}^q |s_i| = l_i$.

Case $\exists$y: We next show how to eliminate an existential set quantifier $\exists y$ from

$$\exists y.\ \exists^+ l_1 \dots l_q.\ \bigwedge_{i=1}^{q} |s_i| = l_i \ \wedge\ G_r \tag{3}$$

which is equivalent to $\exists^+ l_1 \dots l_q.\ (\exists y. \bigwedge_{i=1}^{q} |s_i| = l_i) \ \wedge\ G_r$. This is the key step of the algorithm and relies on the following lemma.

LEMMA 1. *Let $b_1, \dots, b_n$ be finite disjoint sets, and $l_1, \dots, l_n, k_1, \dots, k_n$ be natural numbers. Then the following two statements are equivalent:*

1. *There exists a finite set y such that*

$$\bigwedge_{i=1}^{n} |b_i \cap y| = k_i \ \wedge\ |b_i \cap y^c| = l_i \tag{1}$$

2. $$\bigwedge_{i=1}^{n} |b_i| = k_i + l_i \tag{2}$$

Proof. ($\Rightarrow$) Suppose that there exists a set y satisfying (1). Because $b_i \cap y$ and $b_i \cap y^c$ are disjoint, $|b_i| = |b_i \cap y| + |b_i \cap y^c|$, so $|b_i| = k_i + l_i$.

($\Leftarrow$) Suppose that (2) holds, so $|b_i| = k_i + l_i$ for each of the pairwise disjoint sets $b_1, \dots, b_n$. For each b_i choose a subset $y_i \subseteq b_i$ such that $|y_i| = k_i$. Because $|b_i| = k_i + l_i$, we have $|b_i \cap y_i^c| = l_i$. Having chosen $y_1, \dots, y_n$, let $y = \bigcup_{i=1}^{n} y_i$. For $i \neq j$ we have $b_i \cap y_j = \emptyset$ and $b_i \cap y_j^c = b_i$, so $b_i \cap y = y_i$ and $b_i \cap y^c = b_i \cap y_i^c$. By the choice of y_i, we conclude that y is the desired set for which (1) holds. ∎

In the quantifier elimination step, assume without loss of generality that the set variables $s_1, \dots, s_q$ are numbered such that $s_{2i-1} \equiv s'_i \cap y^c$ and $s_{2i} \equiv s'_i \cap y$ for some cube s'_i. Then apply Lemma 1 and replace each pair of conjuncts

$$|s'_i \cap y^c| = l_{2i-1} \ \wedge\ |s'_i \cap y| = l_{2i}$$

with the conjunct $|s'_i| = l_{2i-1} + l_{2i}$, yielding the formula

$$\exists^+ l_1 \dots l_q.\ \bigwedge_{i=1}^{q'} |s'_i| = l_{2i-1} + l_{2i} \ \wedge\ G_r \tag{4}$$

for $q' = 2^{e-1}$. Finally, to obtain a formula of the form (2) for $r+1$, introduce fresh variables l'_i constrained by $l'_i = l_{2i-1} + l_{2i}$, rewrite (4) as

$$\exists^+ l'_1 \ldots l'_{q'}. \bigwedge_{i=1}^{q'} |s'_i| = l'_i \ \wedge \ (\exists^+ l_1 \ldots l_q. \bigwedge_{i=1}^{q'} l'_i = l_{2i-1} + l_{2i} \ \wedge \ G_r)$$

and let

$$G_{r+1} \equiv \exists^+ l_1 \ldots l_q. \bigwedge_{i=1}^{q'} l'_i = l_{2i-1} + l_{2i} \ \wedge \ G_r \qquad (\exists\text{-step})$$

This completes the description of the elimination of an existential set quantifier $\exists y$.

Case $\forall$y: To eliminate a set quantifier $\forall y$, observe that

$$\neg(\exists^+ l_1 \ldots l_q. \bigwedge_{i=1}^{q} |s_i| = l_i \ \wedge \ G_r)$$

is equivalent to $\exists^+ l_1 \ldots l_q. \bigwedge_{i=1}^{q} |s_i| = l_i \ \wedge \ \neg G_r$, because existential quantifiers over l_i together with the conjuncts $|s_i| = l_i$ act as definitions for l_i, so we may first substitute all values l_i into G_r, then perform the negation, and then extract back the definitions of all values l_i. By expressing $\forall y$ as $\neg\exists y \neg$, we can show that the elimination of $\forall y$ is analogous to elimination of $\exists y$: introduce fresh variables $l'_i = l_{2i-1} + l_{2i}$ and let

$$G_{r+1} \equiv \forall^+ l_1 \ldots l_q. \ (\bigwedge_{i=1}^{q'} l'_i = l_{2i-1} + l_{2i}) \ \Rightarrow \ G_r \qquad (\forall\text{-step})$$

Final step: After eliminating all quantifiers by repeatedly applying the step of the algorithm, we obtain a formula of the form $\exists^+ l. \ |\mathbf{1}| = l \wedge G_{p+1}(l)$. Namely, in the step when we have only one set variable y and its negation y^c, we can write $|y|$ and $|y^c|$ as $|\mathbf{1} \cap y$ and $|\mathbf{1} \cap y^c|$ and apply the algorithm one more time. We then define the result of the algorithm, denoted $\alpha(F_0)$, to be the PA sentence $G_{p+1}(\mathsf{MAXC})$.

We summarize the algorithm in Figure 11. We use $f; g$ to denote the function composition $g \circ f$, and we use f^* to denote iterative application of function f. The prenex function transforms a formula into the prenex form, whereas the separate function transforms it into form (1). We have argued above that each of the individual steps of the algorithm is equivalence preserving, so we have the following lemma.

LEMMA 2. *The transformations* prenex*,* separate*, α_1, α_F are all equivalence preserving (with respect to the* BAPA *interepretation).*

By induction we obtain the correctness of our algorithm.

$$\begin{aligned}
&\alpha_1\,(\exists y.\ \exists^+ l_1 \ldots l_{2q'}.\ \textstyle\bigwedge_{i=1}^{q'} |s_i \cap y^c| = l_{2i-1} \wedge |s_i \cap y| = l_{2i} \wedge G_r) = \\
&\qquad \exists^+ l'_1 \ldots l'_{q'}.\ \textstyle\bigwedge_{i=1}^{q'} |s_i| = l'_i \wedge \exists^+ l_1 \ldots l_{2q'}. \bigwedge_{i=1}^{q'} .l'_i = l_{2i-1} + l_{2i} \wedge G_r \\
&\alpha_1\,(\forall y.\ \exists^+ l_1 \ldots l_{2q'}.\ \textstyle\bigwedge_{i=1}^{q'} |s_i \cap y^c| = l_{2i-1} \wedge |s_i \cap y| = l_{2i} \wedge G_r) = \\
&\qquad \exists^+ l'_1 \ldots l'_{q'}.\ \textstyle\bigwedge_{i=1}^{q'} |s_i| = l'_i \wedge \forall^+ l_1 \ldots l_{2q'}. \bigwedge_{i=1}^{q'} .l'_i = l_{2i-1} + l_{2i} \Rightarrow G_r \\
&\alpha_1\,(\exists k.\ \exists^+ l_1 \ldots l_{2q'}.\ \textstyle\bigwedge_{i=1}^{q'} |s_i \cap y^c| = l_{2i-1} \wedge |s_i \cap y| = l_{2i} \wedge G_r) = \\
&\qquad \exists^+ l_1 \ldots l_{2q'}.\ \textstyle\bigwedge_{i=1}^{q'} |s_i \cap y^c| = l_{2i-1} \wedge |s_i \cap y| = l_{2i} \wedge \exists k. G_r \\
&\alpha_1\,(\forall k.\ \exists^+ l_1 \ldots l_{2q'}.\ \textstyle\bigwedge_{i=1}^{q'} |s_i \cap y^c| = l_{2i-1} \wedge |s_i \cap y| = l_{2i} \wedge G_r) = \\
&\qquad \exists^+ l_1 \ldots l_{2q'}.\ \textstyle\bigwedge_{i=1}^{q'} |s_i \cap y^c| = l_{2i-1} \wedge |s_i \cap y| = l_{2i} \wedge \forall k. G_r
\end{aligned}$$

$$\alpha_F(G(|\mathbf{1}|)) = G(\mathsf{MAXC})$$

$$\alpha = \mathsf{prenex}\ ;\ \mathsf{separate}\ ;\ \alpha_1^*\ ;\ \alpha_F$$

Figure 11. Algorithm α for translating BAPA sentences to PA sentences

THEOREM 3. *The algorithm α in Figure 11 maps each* BAPA-*sentence F_0 into an equivalent* PA-*sentence $\alpha(F_0)$.*

The validity of PA sentences is decidable (Presburger, 1929). In combination with a decision procedure for PA such as (Pugh, 1991; Boigelot et al., 2005; Henriksen et al., 1995), the algorithm α is a decision procedure for BAPA sentences.

4.1. Example Run of Algorithm α

As an illustration, we show the result of runing the algorithm α on the BAPA formula in Figure 6. The result is the PA formula in Figure 12. Note that the structure of the resulting formula mimics the structure of the original formula: every set quantifier is replaced by the corresponding block of quantifiers over non-negative integers constrained to partition the previously introduced integer variables. Figure 13 presents the correspondence between the set variables of the BAPA formula and the integer variables of the translated PA formula. Note that the relationship $\mathsf{content}' = \mathsf{content} \cup e$ translates into the conjunction of the constraints $|\mathsf{content}' \cap (\mathsf{content} \cup e)^c| = 0 \wedge |(\mathsf{content} \cup e) \cap \mathsf{content}'^c| = 0$, which reduces to the conjunction $l_{100} = 0 \wedge l_{011} + l_{001} + l_{010} = 0$ using the translation of set expressions into the disjoint union of partitions, and the correspondence in Figure 13.

$$
\begin{array}{l}
\forall^+ l_1.\forall^+ l_0.\ \mathsf{MAXC} = l_1 + l_0 \Rightarrow \\
\forall^+ l_{11}.\forall^+ l_{01}.\forall^+ l_{10}.\forall^+ l_{00}. \\
l_1 = l_{11} + l_{01} \land l_0 = l_{10} + l_{00} \Rightarrow \\
\quad \forall^+ l_{111}.\ \forall^+ l_{011}.\ \forall^+ l_{101}.\ \forall^+ l_{001}. \\
\quad \forall^+ l_{110}.\ \forall^+ l_{010}.\ \forall^+ l_{100}.\ \forall^+ l_{000}. \\
\quad l_{11} = l_{111} + l_{011} \ \land l_{01} = l_{101} + l_{001} \land \\
\quad l_{10} = l_{110} + l_{010} \ \land l_{00} = l_{100} + l_{000} \Rightarrow \\
\qquad \forall \mathit{size}.\forall \mathit{size}'. \\
\qquad\quad (l_{111} + l_{011} + l_{101} + l_{001} = 1 \land \\
\qquad\quad l_{111} + l_{011} = 0 \land \\
\qquad\quad l_{111} + l_{011} + l_{110} + l_{010} = \mathit{size} \land \\
\qquad\quad l_{100} = 0 \land \\
\qquad\quad l_{011} + l_{001} + l_{010} = 0 \land \\
\qquad\quad \mathit{size}' = \mathit{size} + 1) \Rightarrow \\
\qquad\qquad (0 < \mathit{size}' \land \\
\qquad\qquad l_{111} + l_{101} + l_{110} + l_{100} = \mathit{size}')
\end{array}
$$

Figure 12. The translation of the BAPA sentence from Figure 6 into a PA sentence

general relationship:

$$l_{i_1,\ldots,i_k} = |\mathsf{set}_q^{i_1} \cap \mathsf{set}_{q+1}^{i_2} \cap \ldots \cap \mathsf{set}_S^{i_k}|$$
$$q = S - (k-1)$$
(S is number of set variables)

in this example:

$$\mathsf{set}_1 = \mathsf{content}'$$
$$\mathsf{set}_2 = \mathsf{content}$$
$$\mathsf{set}_3 = e$$

$$
\left.
\begin{array}{lcl}
l_{000} &=& |\mathsf{content}'^c \cap \mathsf{content}^c \cap e^c| \\
l_{001} &=& |\mathsf{content}'^c \cap \mathsf{content}^c \cap e| \\
l_{010} &=& |\mathsf{content}'^c \cap \mathsf{content} \cap e^c| \\
l_{011} &=& |\mathsf{content}'^c \cap \mathsf{content} \cap e| \\
l_{100} &=& |\mathsf{content}' \cap \mathsf{content}^c \cap e^c| \\
l_{101} &=& |\mathsf{content}' \cap \mathsf{content}^c \cap e| \\
l_{110} &=& |\mathsf{content}' \cap \mathsf{content} \cap e^c| \\
l_{111} &=& |\mathsf{content}' \cap \mathsf{content} \cap e|
\end{array}
\right\} (H)
$$

Figure 13. The Correspondence, denoted H, between Integer Variables in Figure 12 and Set Variables in Figure 6

5. Complexity of BAPA

In this section we analyze the algorithm α from Section 4 and use it to show that the computational complexity of BAPA is identical to the complexity of PA, which is $\mathsf{STA}(*, 2^{2^{n^{O(1)}}}, n)$ (Berman, 1980), that is, alternating doubly exponential time with a linear number of alternations.

An alternating Turing machine (Chandra et al., 1981) is a generalization of a non-deterministic Turing machine that, in addition to making non-deterministic (existential) choices, can make universal choices. A universal choice step succeeds iff the computation succeeds for all the chosen values. We then have the following definition.

DEFINITION 4. (Berman, 1980). $\mathsf{STA}(s(n), t(n), a(n))$ *denotes the class of languages computable using an alternating Turing machine that uses $s(n)$ cells on its tape, runs in $t(n)$ steps, and performs $a(n)$ alternations.*

In the remainder of this section we first show the lower bound on the complexity of BAPA. We then use the algorithm α in the previous section and a parameterized upper bound on PA to establish the matching upper bound on BAPA. We finally show that the algorithm α can also decide BA formulas using optimal resources $\mathsf{STA}(*, 2^n, n)$, which is the complexity established in (Kozen, 1980). Moreover, by construction, our procedure reduces to the procedure for PA formulas if there are no set quantifiers. Therefore, the algorithm α can be used as a component for an optimal algorithm for BAPA but also for the special cases of BA and PA.

5.1. Lower Bound on the Complexity of Deciding BAPA

The lower bound of PA follows by showing how to encode full integer arithmetic in which quantifiers are bounded by 2^{2^n} using formulas of size n (Kozen, 2006, Lecture 23).

FACT 1. (Berman, 1980, Page 75). *The truth value of PA formulas of size $O(n)$ can encode the acceptance of an alternating Turing machine with n alternations running in 2^{2^n} steps.*

Because BAPA contains PA, the lower bound directly applies to BAPA.

LEMMA 5. *The truth value of* BAPA *formulas of size $O(n)$ can encode the acceptance of an alternating Turing machine with n alternations running in 2^{2^n} steps.*

5.2. Parameterized Upper Bound on PA

As a step towards establishing the upper bound for BAPA, we show a parameterized upper bound on PA. We use the following result.

FACT 2. (Reddy and Loveland, 1978, Page 324). *If F is a closed* PA *formula $(Q_1 x_1) \ldots (Q_r x_r) G(x_1, \ldots, x_r)$ of size $n > 4$ with m quantifier alternations, where $Q_1, \ldots, Q_r$ are quantifiers and G is quantifier-free, then F is true iff the formula*

$$\left(Q_1 x_1 \leq 2^{2^{(c+1)n^{m+3}}}\right) \ldots \left(Q_r x_r \leq 2^{2^{(c+r)n^{m+3}}}\right) G(x_1, \ldots, x_r)$$

with bounded quantifiers is true for some $c > 0$.

Fact 2 allows us to establish the following parameterized upper bound on PA.

THEOREM 6. *A* PA *sentence of size n with m quantifier alternations can be decided in* $\mathsf{STA}(*, 2^{n^{O(m)}}, m)$.

Proof. Analogous to the proof of the space upper bound in (Reddy and Loveland, 1978, Page 325). To decide formulas with m quantifier alternations, use an alternating Turing machine that does m alternations, mimicking the quantifier alternations. The Turing machine guesses the assignments to variables $x_1, \ldots, x_r$ in ranges given by Fact 2. This requires guessing

$$2^{(c+1)n^{m+3}} + \ldots + 2^{(c+r)n^{m+3}} \leq 2^{c_1 n^{m+4}}$$

bits (because $r \leq n$), and then evaluating the formula in time proportional to $n 2^{c_1 n^{m+4}} \leq 2^{c_2 n^{m+4}}$, which is in $2^{n^{O(m)}}$. ■

5.3. Upper Bound on the Complexity of Deciding BAPA

We next show that the algorithm in Section 4 transforms a BAPA sentence F_0 into a PA sentence whose size is at most exponential and which has the same number of quantifier alternations. Theorem 6 will then give us the upper bound on BAPA that matches the lower bound of Lemma 5.

If F is a formula in prenex form, let $\mathsf{size}(F)$ denote the size of F, and let $\mathsf{alts}(F)$ denote the number of quantifier alternations in F.

LEMMA 7. *For the algorithm α from Section 4 there is a constant $c > 0$ such that $\mathsf{size}(\alpha(F_0)) \leq 2^{c \cdot \mathsf{size}(F_0)}$ and $\mathsf{alts}(\alpha(F_0)) = \mathsf{alts}(F_0)$. Moreover, the algorithm α runs in $2^{O(\mathsf{size}(F_0))}$ deterministic time.*

Proof. To gain some intuition on the size of $\alpha(F_0)$ compared to the size of F_0, compare first the formula in Figure 12 with the original formula in Figure 6. Let n denote the size of the initial formula F_0 and let

S be the number of set variables. Note that the following operations are polynomially bounded in time and space: 1) transforming a formula into prenex form, 2) transforming relations $b_1 = b_2$ and $b_1 \subseteq b_2$ into the form $|b| = 0$. Introducing set variables for each partition and replacing each $|b|$ with a sum of integer variables yields formula G_1 whose size is bounded by $O(n2^S S)$ (the last S factor is because representing a variable from the set of K variables requires space $\log K$). The subsequent transformations introduce the existing integer quantifiers, whose size is bounded by n, and introduce additionally $2^{S-1} + \ldots + 2 + 1 = 2^S - 1$ new integer variables along with the equations that define them. Note that the defining equations always have the form $l'_i = l_{2i-1} + l_{2i}$ and have size bounded by S. We therefore conclude that the size of $\alpha(F_0)$ is $O(nS(2^S + 2^S))$ and therefore $O(nS2^S)$, which is in $2^{O(n)}$. Note that we have obtained a more precise bound $O(nS2^S)$ indicating that the exponential explosion is caused only by set variables. Finally, the fact that the number of quantifier alternations is the same in F_0 and $\alpha(F_0)$ is immediate because the algorithm replaces one set quantifier with a block of corresponding integer quantifiers. ∎

Combining Lemma 7 with Theorem 6 we obtain the upper bound for BAPA.

LEMMA 8. *The validity of a* BAPA *sentence of size n and the number of quantifier alternations m can be decided in* $\mathsf{STA}(*, 2^{2^{O(mn)}}, m)$.

Proof. The algorithm first applies the algorithm α and then applies the algorithm from Theorem 6. Consider a BAPA formula of size n with m quantifier alternations. By Lemma 7 applying α takes $2^{O(n)}$ steps and produces a formula of size $2^{O(n)}$ with m quantifier alternations. Theorem 6 then allows deciding such formula in $\mathsf{STA}(*, 2^{2^{(2^{O(n)})^{O(m)}}}, m)$, which is $\mathsf{STA}(*, 2^{2^{O(mn)}}, m)$. ∎

Summarizing Lemma 5 and Lemma 8, taking into account that $2^{O(mn)}$ is in $2^{n^{O(1)}}$ for $m \leq n$, we establish the desired theorem.

THEOREM 9. *The validity of* BAPA *formulas is* $\mathsf{STA}(*, 2^{2^{n^{O(1)}}}, n)$-*complete.*

5.4. Deciding BA as a Special Case of BAPA

We next analyze the result of applying the algorithm α to a pure BA sentence F_0. By a pure BA sentence we mean a BA sentence without cardinality constraints, containing only the standard operations $\cap, \cup, {}^c$ and the relations $\subseteq, =$. At first, it might seem that the algorithm α

is not a reasonable approach to deciding BA formulas given that the complexity of PA is worse than the corresponding of BA. However, we show that the sentences $\mathsf{PA}_{\mathsf{BA}} = \{\alpha(F_0) \mid F_0 \text{ is in } \mathsf{BA}\}$ generated by applying α to pure BA sentences can be decided using resources optimal for BA (Kozen, 1980). The key observation is that if a PA formula is in $\mathsf{PA}_{\mathsf{BA}}$, then we can use the bounds for quantified variables that are smaller than the bounds in Fact 2.

Let F_0 be a pure BA formula and let S be the number of set variables in F_0 (the set variables are the only variables in F_0). Let $l_1, \ldots, l_q$ be the free variables of the formula $G_r(l_1, \ldots, l_q)$ in the algorithm α. Then $q = 2^e$ for $e = S+1-r$. Let $w_1, \ldots, w_q$ be integers specifying the values of $l_1, \ldots, l_q$. We then have the following lemma.

LEMMA 10. *For each r where $1 \leq r \leq S$, formula $G_r(w_1, \ldots, w_q)$ is equivalent to formula $G_r(\bar{w}_1, \ldots, \bar{w}_q)$ where $\bar{w}_i = \min(w_i, 2^{r-1})$.*

Proof. We prove the claim by induction. For $r = 1$, observe that the translation of a quantifier-free part of the pure BA formula yields a PA formula F_1 whose all atomic formulas are of the form $l_{i_1} + \ldots + l_{i_k} = 0$, which are equivalent to $\bigvee_{j=1}^{k} l_{i_j} = 0$. Therefore, the truth-value of F_1 depends only on whether the integer variables are zero or non-zero, which means that we may restrict the variables to interval $[0, 1]$.

For the inductive step, consider the elimination of a set variable, and assume that the property holds for G_r and for all q-tuples of non-negative integers $w_1, \ldots, w_q$. Let $q' = q/2$ and $w'_1, \ldots, w'_{q'}$ be a tuple of non-negative integers. We show that $G_{r+1}(w'_1, \ldots, w'_{q'})$ is equivalent to $G_{r+1}(\bar{w}'_1, \ldots, \bar{w}'_{q'})$. We consider the case when G_{r+1} is obtained by eliminating an existential set quantifier, so

$$G_{r+1} \equiv \exists^+ l_1 \ldots l_q.\ \bigwedge_{i=1}^{q'} l'_i = l_{2i-1} + l_{2i} \ \wedge\ G_r$$

The case for the universal quantifier can be obtained from the proof for the existential one by expressing $\forall$ as $\neg\exists\neg$. We show both directions of the equivalence.

Suppose first that $G_{r+1}(\bar{w}'_1, \ldots, \bar{w}'_{q'})$ holds. Then for each $\bar{w}'_i$ there are u_{2i-1} and u_{2i} such that $\bar{w}'_i = u_{2i-1} + u_{2i}$ and $G_r(u_1, \ldots, u_q)$. We define the witnesses $w_1, \ldots, w_q$ for existential quantifiers as follows. If $w'_i \leq 2^r$, then $\bar{w}'_i = w'_i$, so we use the same witnesses, letting $w_{2i-1} = u_{2i-1}$ and $w_{2i} = u_{2i}$. Let $w'_i > 2^r$, then $\bar{w}'_i = 2^r$. Because $u_{2i-1} + u_{2i} = 2^r$, we have $u_{2i-1} \geq 2^{r-1}$ or $u_{2i} \geq 2^{r-1}$. Suppose, without loss of generality, $u_{2i} \geq 2^{r-1}$. Then let $w_{2i-1} = u_{2i-1}$ and let $w_{2i} = w'_i - u_{2i-1}$. Because $w_{2i} \geq 2^{r-1}$ and $u_{2i} \geq 2^{r-1}$, by induction hypothesis we have

$$G_r(\ldots, w_{2i}, \ldots) \iff G_r(\ldots, u_{2i}, \ldots) \iff G_r(\ldots, 2^{r-1}, \ldots)$$

For $w_1, \ldots, w_q$ chosen as above we therefore have $w'_i = w_{2i-1} + w_{2i}$ and $G_r(w_1, \ldots, w_q)$, which by definition of G_{r+1} means that $G_{r+1}(w'_1, \ldots, w'_{q'})$ holds.

Conversely, suppose that $G_{r+1}(w'_1, \ldots, w'_{q'})$ holds. Then there are $w_1, \ldots, w_q$ such that $G_r(w_1, \ldots, w_q)$ and $w'_i = w_{2i-1} + w_{2i}$. If $w_{2i-1} \leq 2^{r-1}$ and $w_{2i} \leq 2^{r-1}$ then $w'_i \leq 2^r$ so let $u_{2i-1} = w_{2i-1}$ and $u_{2i} = w_{2i}$. If $w_{2i-1} > 2^{r-1}$ and $w_{2i} > 2^{r-1}$ then let $u_{2i-1} = 2^{r-1}$ and $u_{2i} = 2^{r-1}$; we have $u_{2i-1} + u_{2i} = 2^r = \bar{w}'_i$ because $w'_i > 2^r$. If $w_{2i-1} > 2^{r-1}$ and $w_{2i} \leq 2^{r-1}$ then let $u_{2i-1} = 2^r - w_{2i}$ and $u_{2i} = w_{2i}$. By induction hypothesis we have $G_r(u_1, \ldots, u_q) \iff G_r(w_1, \ldots, w_q)$. Furthermore, $u_{2i-1} + u_{2i} = \bar{w}'_i$, so $G_{r+1}(\bar{w}'_1, \ldots, \bar{w}'_{q'})$ by definition of G_{r+1}. ∎

THEOREM 11. *The decision problem for* $\mathsf{PA}_{\mathsf{BA}}$ *is in* $\mathsf{STA}(*, 2^{O(n)}, n)$.

Proof. Consider a formula F_0 of size n with S variables. Then $\alpha(F_0) = G_{S+1}$. By Lemma 7, $\mathsf{size}(\alpha(F_0))$ is in $2^{O(n)}$ and $\alpha(F_0)$ has at most S quantifier alternations. By Lemma 10, it suffices for the outermost quantified variable of $\alpha(F_0)$ to range over the integer interval $[0, 2^S]$, and the range of subsequent variables is even smaller. Therefore, the value of each of the $2^{O(S)}$ variables is given by $O(S)$ bits. The values of all bound variables in $\alpha(F_0)$ can therefore be guessed in alternating time $2^{O(S)}$ using S alternations. The truth value of a PA formula for given values of variables can be evaluated in time polynomial in the size of the formula, so deciding $\alpha(F_0)$ can be done in $\mathsf{STA}(*, 2^{O(n)}, S)$, which is bounded by $\mathsf{STA}(*, 2^{O(n)}, n)$. ∎

6. Eliminating Individual Variables from a Formula

Section 4 described an algorithm for BAPA that reduces all set quantifiers in a BAPA sentence to integer quantifiers. The advantage of such an algorithm is that it can be applied to combinations of BA with extensions of PA that need not support quantifier elimination (see Section 8.2). Moreover, this version of the algorithm made the complexity analysis in Section 5 easier. However, as we discussed in Section 3.4, there are uses for an algorithm that eliminates a quantified variable from a formula with free variables, yielding an equivalent quantifier-free formula. In this section we explain that the individual step α_1 of the algorithm α can be used for this purpose.

Quantifier elimination based on our algorithm is possible because PA itself admits quantifier elimination. Therefore, after transforming a set quantifier into an integer quantifier, we can remove the resulting integer quantifier and substitute back the variables constrained by $l_i = |s_i|$. Denote this elimination of integer quantifiers by PAelim. Then the algorithm α', given by

$$\mathsf{separate}\ ;\ \alpha_1\ ;\ \mathsf{PAelim}$$

eliminates one set or integer quantifier from a BAPA formula $Qv.F$, even if F contains free variables (see also (Kuncak and Rinard, 2004) for details). Lemma 2 again implies the correctness of this approach.

Example. We illustrate the algorithm α' on the example in Figure 10. We use the naming convention given by the formula H for cardinalities of Venn regions from Figure 13. After applying $\mathsf{separate}$ we obtain the formula

$$\begin{array}{l} \exists e.\ \exists^+ l_{000}, \ldots, l_{111}.\ H \ \land \\ \quad l_{111} + l_{011} + l_{101} + l_{001} = 1 \ \land \\ \quad l_{100} = 0 \ \land\ l_{011} + l_{001} + l_{010} = 0 \end{array}$$

After applying α_1 we obtain the formula

$$\begin{array}{l} \exists l_{00}, l_{01}, l_{10}, l_{11}.\ l_{00} = |\mathsf{content}'^c \cap \mathsf{content}^c| \ \land\ l_{01} = |\mathsf{content}'^c \cap \mathsf{content}| \ \land \\ \qquad\qquad l_{10} = |\mathsf{content}' \cap \mathsf{content}^c| \land\ l_{11} = |\mathsf{content}' \cap \mathsf{content}| \ \land\ G \end{array}$$

where G is the PA formula

$$\begin{array}{ll} \exists^+ l_{000}, \ldots, l_{111}. & l_{00} = l_{000} + l_{001} \ \land l_{01} = l_{010} + l_{011} \ \land \\ & l_{10} = l_{100} + l_{101} \ \land l_{11} = l_{110} + l_{111} \ \land \\ & l_{111} + l_{011} + l_{101} + l_{001} = 1 \ \land \\ & l_{100} = 0 \ \land\ l_{011} + l_{001} + l_{010} = 0 \end{array}$$

Applying quantifier elimination for PA and simplifications of the quantifier-free formula, we reduce G to

$$l_{01} = 0 \ \land\ l_{10} \leq 1 \ \land\ l_{11} + l_{10} \geq 1$$

After substituting back the definitions of $l_{00}, \ldots, l_{11}$ we obtain

$$\begin{array}{l} |\mathsf{content}'^c \cap \mathsf{content}| = 0 \ \land\ |\mathsf{content}' \cap \mathsf{content}^c| \leq 1 \land \\ |\mathsf{content}' \cap \mathsf{content}| + |\mathsf{content}' \cap \mathsf{content}^c| \geq 1 \end{array}$$

which can indeed be simplified to the result in Figure 10.

6.1. Reducing the Number of Integer Variables

The approaches for deciding BAPA described so far always introduce 2^S integer variables where S is the number of set variables in the formula. We next describe observations that, although not an improvement in the worst case, may be helpful for certain classes of formulas.

First, as pointed out in (Ohlbach and Koehler, 1998), if the input formula entails any BA identities (which can be represented as $|b| = 0$), then the number of non-empty Venn regions decreases, which reduces the number of integer variables in the resulting PA formula, and eliminates such atomic formulas b from further considerations.

Second, when eliminating a particular variable y, we can avoid considering Venn regions with respect to all variables of the formula, and only consider those variables x for which there exists an expression $b_i(x, y)$ where both x and y occur. This requires using a version $\mathsf{separate}_0$ of separation that introduces integer variables only for those terms $|b|$ that occur in the BAPA formula that results from reducing any remaining BA atomic formulas to the form $|b| = 0$. Like the form (1) obtained by separate in Section 4, the result of $\mathsf{separate}_0$ contains a conjunction of formulas $|b_i| = l_i$ with a PA formula, with two important differences 1) in the case of $\mathsf{separate}_0$ the resulting formula is polynomial in the original size and 2) b_i are arbitrary BA expressions as opposed to Venn regions. Let $a_1(y), \ldots, a_q(y)$ be those terms b_i that contain y, and let $x_1, \ldots, x_{S_1}$ be the free variables in $a_1(y), \ldots, a_q(y)$. When eliminating quantifier Qy, it suffices to introduce 2^{S_1} integer variables corresponding to the the partitions with respect to $x_1, \ldots, x_{S_1}$, which may be an improvement because $S_1 \leq S$.

The final observation is useful if the number q of terms $a_1(y), \ldots, a_q(y)$ satisfies the property $2q < S_1$, i.e. there is a large number of variables, but a small number of BA terms containing them. In this case, consider all Boolean combinations $t_1, \ldots, t_u$ of the $2q$ expressions $a_1(\mathbf{0}), a_1(\mathbf{1}), a_2(\mathbf{0}), a_2(\mathbf{1}), \ldots, a_q(\mathbf{0}), a_q(\mathbf{1})$. For each a_i, we have

$$a_i(y) = (y \cap a_i(\mathbf{0})) \cup (y^c \cap a_i(\mathbf{1}))$$

Each $a_i(\mathbf{0})$ and each $a_i(\mathbf{1})$ is a disjoint union of cubes over the BA expressions $t_1, \ldots, t_u$, so each $a_i(y)$ is a disjoint union of cubes over the BA expressions y, $t_1, \ldots, t_u$. It therefore suffices to introduce 2^{2q} integer variables denoting all terms of the form $y \cap t_i$ and $y^c \cap t_i$, as opposed to 2^{S_1} integer variables.

7. Experience Using Our Decision Procedure for BAPA

Our initial experience with BAPA is in the context of the Jahob system (Kuncak and Rinard, 2006) for verifying data structure consistency of Java programs. Jahob parses Java source code annotated with formulas in Isabelle syntax written in comments, generates verification conditions, and uses decision procedures and theorem provers to discharge these verification conditions. Jahob contains interfaces to the interactive theorem provers, first-order theorem provers, Nelson-Oppen style theorem provers, as well as the Omega Calculator (Pugh, 1991) and the LASH (Boigelot et al., 2005) decision procedures for PA.

Using Jahob, we generated verification conditions for several Java program fragments that require reasoning about sets and their cardinalities, for example, to prove the equality between the set representing the number of elements in a list and the integer field size after they have been updated, as well as several other examples from Section 3. We found that the existing automated techniques were able to deal with some of the formulas involving only sets or only integers, but not with the formulas that relate cardinalities of operations on sets to the cardinalities of the individual sets. These formulas can be proved in Isabelle, but require user interaction in terms of auxiliary lemmas. On the other hand, our implementation of the decision procedure automatically discharges these formulas.

Our initial experience indicates that a straightforward implementation of the algorithm α works fast as long as the number of set variables is small; typical timings are fractions of a second for 4 or less set variables and less than 10 seconds for 5 variables. More than 5 set variables cause the PA decision procedure to run out of memory. (We used the Omega Calculator to decide PA formulas because we found that it outperforms LASH on the formulas generated from our examples.) On the other hand, the decision procedure is much less sensitive to the number of integer variables in BAPA formulas, because they translate into the same number of integer variables in the generated PA formula.

Our current implementation makes use of certain formula transformations to reduce the size of the generated PA formula. We found that eliminating set variables by substitution of equals for equals is an effective optimization. We also observed that lifting quantifiers to the top level noticeably improves the performance of the Omega Calculator. These transformations extend the range of formulas that the current implementation can handle. A possible alternative to the current approach is to interleave the elimination of integer variables with the elimination of the set variables, and to perform formula simplifications during this process. Finally, once we obtain a formula with only existential quantifiers, we can decide its satisfiability more efficiently using the recent improvements for quantifier-free formulas (Section 8.3).

8. Further Observations

We next sketch some further observations about BAPA.

8.1. BAPA of Countably Infinite Sets

Note that our results also extend to the generalization of BAPA where set variables range over subsets of an arbitrary (not necessarily finite) set, which follows from the decidability of the first-order theory of the addition of cardinals (Feferman and Vaught, 1959, Page 78); see (Zarba, 2005, Appendix A) for the complexity of the quantifier-free case. For simplicity of formula semantics, we here consider only the case of all subsets of a countable set, and argue that the complexity results we have developed above for the finite sets still apply. We first generalize the language of BAPA and the interpretation of BAPA operations, as follows. Introduce a function $\mathsf{inf}(b)$ which returns 0 if b is a finite set and 1 if b is an infinite set. Define $|b|$ to be some arbitrary integer (for concreteness, zero) if b is infinite, and the cardinality of b if b is finite. A countable or a finite cardinal can therefore be represented in PA using a pair (k, i) of an integer k and an infinity flag i, where we put $k = 0$ when $i = 1$. The relation representing the addition of cardinals $(k_1, i_1) + (k_2, i_2) = (k_3, i_3)$ is then definable by formula

$$(i_1 = 0 \wedge i_2 = 0 \wedge i_3 = 0 \wedge k_1 + k_2 = k_3) \ \vee \ ((i_1 = 1 \vee i_2 = 1) \wedge i_3 = 1 \wedge k_3 = 0)$$

Note that $\mathsf{inf}(x) = \max(\mathsf{inf}(x \cap y),\ \mathsf{inf}(x \cap y^c))$, which allows the reduction of all occurrences of $\mathsf{inf}(b)$ expressions to occurrences where b is a Venn region. Moreover, we have the following generalization of Lemma 1.

LEMMA 12. *Let $b_1, \ldots, b_n$ be disjoint sets, $l_1, \ldots, l_n, k_1, \ldots, k_n$ be natural numbers, and $p_1, \ldots, p_n, q_1, \ldots, q_n \in \{0, 1\}$ for $1 \leq i \leq n$. Then the following two statements are equivalent:*

1. *There exists a set y such that*

$$\bigwedge_{i=1}^{n} |b_i \cap y| = k_i \wedge \mathsf{inf}(b_i \cap y) = p_i \ \wedge \ |b_i \cap y^c| = l_i \wedge \mathsf{inf}(b_i \cap y^c) = q_i$$

2.

$$\bigwedge_{i=1}^{n} \begin{pmatrix} (p_i = 0 \wedge q_i = 0 \Rightarrow |b_i| = k_i + l_i) \ \wedge \\ (\mathsf{inf}(b_i) = 0 \Leftrightarrow (p_i = 0 \wedge q_i = 0)) \ \wedge \\ (p_i = 1 \Rightarrow k_i = 0) \ \wedge \ (q_i = 1 \Rightarrow l_i = 0) \end{pmatrix}$$

Proof. The case when $p_i = 0$ and $q_i = 0$ follows as in the proof of Lemma 1. When $p_i = 1$ or $q_i = 1$ then b_i contains an infinite set as a subset, so it must be infinite. Conversely, an infinite set can always be split into a set of the desired size and another infinite set, or into two infinite sets. ■

The BAPA decision procedure for the case of countable set then uses Lemma 12 and generalizes the algorithm α in a natural way. The resulting PA formulas are at most polynomially larger than for the case of finite sets, so we obtain a generalization of Theorem 9 to subsets of a countable set.

8.2. BAPA and MSOL over Strings

The weak monadic second-order logic (MSOL) over strings is a decidable logic (Thatcher and Wright, 1968; Henriksen et al., 1995) that can encode Presburger arithmetic by encoding addition using one successor symbol and quantification over sets of elements. There are two important differences between MSOL over strings and BAPA: (1) BAPA can express relationships of the form $|A| = k$ where A is a set variable and k is an integer variable; such relation is not definable in MSOL over strings; (2) when MSOL over strings is used to represent PA operations, the sets contain binary integer digits whereas in BAPA the sets contain uninterpreted elements. Note also that MSOL extended with a construct that takes a set of elements and returns an encoding of the size of that set is undecidable, because it could express MSOL with equicardinality, which is undecidable by a reduction from Post correspondence problem. Despite this difference, the algorithm α gives a way to combine MSOL over strings with BA yielding a decidable theory. Namely, α does not impose any upper bound on the complexity of the theory for reasoning about integers. Therefore, α can decide an extension of BAPA where the constraints on cardinalities of sets are expressed using relations on integers definable in MSOL over strings; these relations go beyond PA (Thomas, 1997, Page 400), (Bruyére et al., 1994).

8.3. Quantifier-Free Fragment of BAPA

The consequence of the quantifier elimination property of BAPA is that the formulas in the quantifier-free fragment of BAPA, denoted QFBAPA, define the same class of relations as BAPA formulas (even though QFBAPA formulas may be substantially larger than the corresponding quantified BAPA formulas). It is therefore interesting to consider the complexity of the satisfiability problem for QFBAPA. Existing algorithms for QFBAPA (Zarba, 2005; Ohlbach and Koehler, 1998) run in non-deterministic exponential time. This is the same time bound obtained by running our algorithm α to produce exponentially larger quantifier-free PA formula, guessing the truth values of conjuncts and then using a non-deterministic polynomial time algorithm to solve the resulting integer linear programming problem (Papadimitriou, 1981). (Quantifier-free PA formulas can be solved using implementations such as CVC Lite (Barrett and Berezin, 2004) and UCLID (Lahiri and Seshia, 2004).) Therefore, the worst-case complexity of α is no worse than that of previously known algorithms.

Is it possible to decide QFBAPA formulas in less than non-deterministic exponential time? Recently, as part of an ongoing work, the first author of the present paper has shown that QFBAPA is solvable in polynomial space. A version of these results is described in (Marnette et al., 2005, Section 3). The result proves the following small model property implying that it suffices to consider cardinalities of sets that are at most exponential in formula size.

THEOREM 13. *Let F be a* QFBAPA *formula of size n. Then F has a solution if and only if it has a solution where all sets are subsets of a set of cardinality $2^{n^{O(1)}}$, and all integer variables are bounded by $2^{n^{O(1)}}$.*

Theorem 13 allows us to reduce, in non-deterministic polynomial time, the satisfiability of any QFBAPA formula to the satisfiability of the conjunction of equations of the form

$$|b_1| = K_1 \ \wedge\ \ldots\ \wedge\ |b_n| = K_n$$

where $b_1, \ldots, b_n$ are Boolean algebra expressions, and $K_1, \ldots, K_n$ are binary-encoded non-negative constants. We call such constraint CBAC constraints (Conjunction of Boolean Algebra expressions with Cardinalities). In (Marnette et al., 2005, Section 3) we present a polynomial-space algorithm that checks the satisfiability of CBAC constraints without ever simultaneously introducing integer variables for all regions in the Venn diagram. This yields a polynomial-space algorithm for QFBAPA. A polynomial space algorithm means the ability to use QBF solvers (Zhang and Malik, 2002; Cadoli et al., 2002) to decide QFBAPA, and can be a significant improvement over non-deterministic exponential time algorithm. However, there is no indication that QFBAPA is PSPACE-hard or even coNP-hard: our preliminary experiments on studying the structure of solutions of randomly generated CBAC constraints suggest that satisfiable constraints have solutions where only a small number q of Venn regions is non-empty. This suggests an algorithm that selects a subset of q Venn region cardinalities and generates an integer linear programming problem where the remaining $2^n - q$ variables are assumed to be zero. A computable polynomial upper bound on q would yield an NP algorithm for QFBAPA.

9. Related Work

Our paper is the first result that shows a complexity bound for the first-order theory of BAPA. A preliminary version of this paper appears in (Kuncak et al., 2005; Kuncak and Rinard, 2004). The decidability for BAPA, presented as BA with equicardinality constraints was shown in (Feferman and Vaught, 1959), see Section 2. A decision procedure for a special case of BAPA was presented in (Zarba, 2004c), which allows only quantification over *elements* but not over *sets* of elements. (Ohlbach and Koehler, 1998) also examine quantifier-free formulas and show how to combine quantifier-free BA constraints with additional constraints using "bridging functions". Bridging functions satisfy homomorphism-like properties and generalize the cardinality operator; we expect that the quantifier-elimination techniques of our paper can be generalized in this direction as well. (Revesz, 2004) presents quantifier elimination and the decidability of a single-sorted version of BAPA that only contains the set sort. Note that bound integer variables can be simulated using bound set variables, but there are notational and clear efficiency reasons to allow integer variables in BAPA.

Presburger arithmetic. The original result on decidability of PA is (Presburger, 1929). The space bound for PA was shown in (Ferrante and Rackoff, 1979). The matching lower and upper bounds for PA were shown in (Berman, 1980), see also (Kozen, 1980, Lecture 24). An analysis parameterized by the number of quantifier alternations is presented in (Reddy and Loveland, 1978). Our implementation uses quantifer-elimination based Omega test (Pugh, 1991). Among the decision procedures for full PA, (Chaieb and Nipkow, 2003) is the only proof-generating version, and is based on (Cooper, 1972). Decidable fragments of arithmetic that go beyond PA are described in (Bozga and Iosif, 2005; Bruyére et al., 1994).

Reasoning about Sets. The first results on decidability of BA of sets are from (Loewenheim, 1915), (Ackermann, 1954, Chapter 4) and use quantifier elimination, from which one can derive small model property. (Kozen, 1980) gives the complexity of the satisfiability problem for arbitrary BA. (Martin and Nipkow, 1989) study unification in Boolean rings. The quantifier-free fragment of BA is shown NP-complete in (Marriott and Odersky, 1994); see (Kuncak and Rinard, 2005) for a generalization of this result using the parameterized complexity of the

Bernays-Schönfinkel-Ramsey class of first-order logic (Börger et al., 1997, Page 258). (Cantone et al., 2001) gives an overview of several fragments of set theory including theories with quantifiers but no cardinality constraints and theories with cardinality constraints but no quantification over sets. The decision procedure for quantifier-free fragment with cardinalities in (Cantone et al., 2001, Chapter 11) introduces exponentially many integer variables to reduce the problem to PA. Among the systems for interactively reasoning about richer theories of sets are Isabelle (Nipkow et al., 2002), HOL (Gordon and Melham, 1993), PVS (Owre et al., 1992). First-order frameworks such as Athena (Arkoudas et al., 2004) can use axiomatizations of sets along with calls to resolution-based theorem provers (Voronkov, 1995; Weidenbach, 2001) to reason about sets.

Combinations of Decidable Theories. The techniques for combining *quantifier-free* theories (Nelson and Oppen, 1979; Ruess and Shankar, 2001) and their generalizations such as (**?**; Zarba, 2004a; Zarba, 2005; Zarba, 2004b; Tiwari, 2000) are of great importance for program verification. The present paper focuses on quantified formulas, which add additional expressive power in writing concise specifications. Among the general results for quantified formulas are the Feferman-Vaught theorem for products (Feferman and Vaught, 1959) and term powers (Kuncak and Rinard, 2003a; Kuncak and Rinard, 2003b). Description logics (Baader et al., 2003) also support sets with cardinalities as well as relations, but do not support quantification over sets. While we have found quantifier elimination to be useful, many problems can be encoded in quantifier-free formulas, which motivates the ongoing work in Section 8.3.

Analyses of Linked Data Structures. In addition to the new technical results, one of the contributions of our paper is to identify the uses of our decision procedure for verifying data structure consistency. We have shown how BAPA enables the verification tools to reason about sets and their sizes. This capability is particularly important for analyses that handle dynamically allocated data structures where the number of objects is statically unbounded (Møller and Schwartzbach, 2001; Yorsh et al., 2004; Sagiv et al., 2002). Recently, these approaches were extended to handle the combinations of the constraints representing data structure contents and constraints representing numerical properties of data structures (Rugina, 2004; Chin et al., 2003). Our result provides a systematic mechanism for building precise and predictable versions of such analyses. Among other constraints used for data structure analysis, BAPA is unique in being a complete algorithm for an expressive theory that supports arbitrary quantifiers.

10. Conclusion

Motivated by static analysis and verification of relations between data structure content and size, we have presented an algorithm for deciding the first-order theory of Boolean algebra with Presburger arithmetic (BAPA), established the precise complexity of BAPA, showing that it is identical to the complexity of PA, implemented the algorithm and applied it to discharge verification conditions. Our experience indicates that the algorithm is useful as a component of a data structure verification system. Furthermore, we expect that the future study of quantifier-free fragments of BAPA and its variations with non-integer measures of sets might be of broader interest in the context of constraint solving and complexity.

Acknowledgements

After writing (Kuncak and Rinard, 2004), we have received substantial useful feedback: we thank Alexis Bes, who pointed out that (Feferman and Vaught, 1959) establishes the decidability of BAPA; Dexter Kozen, who pointed out that (Berman, 1980) proves the precise complexity of PA that can be potentially applied to BAPA, thus improving the space bound we established in (Kuncak et al., 2005; Kuncak and Rinard, 2004); Peter Revesz, who pointed us out to his description of the quantifier-elimination process (Revesz, 2004); and Viorica Sofronie-Stokkermans, who pointed us to (Ohlbach and Koehler, 1998). We also thank Chin Wei-Ngan, Andreas Podelski, Bruno Courcelle, as well as the reviewers of CADE-20 and JAR for useful feedback.

References

Ackermann, W.: 1954, *Solvable Cases of the Decision Problem.* North Holland.

Arkoudas, K., K. Zee, V. Kuncak, and M. Rinard: 2004, 'Verifying a File System Implementation'. In: *Sixth International Conference on Formal Engineering Methods (ICFEM'04)*, Vol. 3308 of *LNCS*. Seattle.

Baader, F., D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider (eds.): 2003, *The Description Logic Handbook: Theory, Implementation and Applications.* CUP.

Barrett, C. and S. Berezin: 2004, 'CVC Lite: A New Implementation of the Cooperating Validity Checker'. In: *Proc.* 16^{th} *Int. Conf. on Computer Aided Verification (CAV '04)*, Vol. 3114 of *Lecture Notes in Computer Science*. pp. 515–518.

Berman, L.: 1980, 'The Complexity of Logical Theories'. *Theoretical Computer Science* **11**(1), 71–77.

Boigelot, B., S. Jodogne, and P. Wolper: 2005, 'An Effective Decision Procedure for Linear Arithmetic Over the Integers and Reals'. *ACM Trans. Comput. Logic* **6**(3), 614–633.

Börger, E., E. Grädel, and Y. Gurevich: 1997, *The Classical Decision Problem*. Springer-Verlag.

Bozga, M. and R. Iosif: 2005, 'On Decidability within the Arithmetic of Addition and Divisibility'. In: *FOSSACS'05*.

Bruyére, V., G. Hansel, C. Michaux, and R. Villemaire: 1994, 'Logic and p-recognizable sets of integers'. *Bull. Belg. Math. Soc. Simon Stevin* **1**, 191–238.

Bultan, T., R. Gerber, and W. Pugh: 1999, 'Model-Checking Concurrent Systems with Unbounded Integer Variables: symbolic representations, approximations, and experimental results'. *ACM Trans. Program. Lang. Syst.* **21**(4), 747–789.

Cadoli, M., M. Schaerf, A. Giovanardi, and M. Giovanardi: 2002, 'An Algorithm to Evaluate Quantified Boolean Formulae and Its Experimental Evaluation.'. *J. Autom. Reasoning* **28**(2), 101–142.

Cantone, D., E. Omodeo, and A. Policriti: 2001, *Set Theory for Computing*. Springer.

Chaieb, A. and T. Nipkow: 2003, 'Generic proof synthesis for Presburger arithmetic'. Technical report, Technische Universität München.

Chandra, A. K., D. C. Kozen, and L. J. Stockmeyer: 1981, 'Alternation'. *J. ACM* **28**(1), 114–133.

Chin, W.-N., S.-C. Khoo, and D. N. Xu: 2003, 'Extending Sized Types with with Collection Analysis'. In: *ACM PEPM'03*.

Cooper, D. C.: 1972, 'Theorem Proving in Arithmetic without Multiplication'. In: B. Meltzer and D. Michie (eds.): *Machine Intelligence*, Vol. 7. Edinburgh University Press, pp. 91–100.

Dewar, R. K.: 1979, 'Programming by Refinement, as Exemplified by the SETL Representation Sublanguage'. *ACM TOPLAS*.

Feferman, S. and R. L. Vaught: 1959, 'The First Order Properties of Products of Algebraic Systems'. *Fundamenta Mathematicae* **47**, 57–103.

Ferrante, J. and C. W. Rackoff: 1979, *The Computational Complexity of Logical Theories*, Vol. 718 of *Lecture Notes in Mathematics*. Springer-Verlag.

Gordon, M. J. C. and T. F. Melham: 1993, *Introduction to HOL, a theorem proving environment for higher-order logic*. Cambridge, England: Cambridge University Press.

Henriksen, J., J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm: 1995, 'Mona: Monadic Second-order logic in practice'. In: *TACAS '95, LNCS 1019*.

Hoare, C. A. R.: 1969, 'An Axiomatic Basis for Computer Programming'. *Communications of the ACM* **12**(10), 576–580.

Kapur, D.: 2004, 'Automatically Generating Loop Invariants using Quantifier Elimination'. In: *IMACS Intl. Conf. on Applications of Computer Algebra*.

Klarlund, N., A. Møller, and M. I. Schwartzbach: 2000, 'MONA Implementation Secrets'. In: *Proc. 5th International Conference on Implementation and Application of Automata*.

Kozen, D.: 1980, 'Complexity of Boolean Algebras'. *Theoretical Computer Science* **10**, 221–247.

Kozen, D.: 2006, *Theory of Computation*. Springer.

Kuncak, V., H. H. Nguyen, and M. Rinard: 2005, 'An Algorithm for Deciding BAPA: Boolean Algebra with Presburger Arithmetic'. In: *20th International Conference on Automated Deduction, CADE-20*. Tallinn, Estonia.

Kuncak, V. and M. Rinard: 2003a, 'On the Theory of Structural Subtyping'. Technical Report 879, Laboratory for Computer Science, Massachusetts Institute of Technology.

Kuncak, V. and M. Rinard: 2003b, 'Structural Subtyping of Non-Recursive Types is Decidable'. In: *Eighteenth Annual IEEE Symposium on Logic in Computer Science*.

Kuncak, V. and M. Rinard: 2004, 'The First-Order Theory of Sets with Cardinality Constraints is Decidable'. Technical Report 958, MIT CSAIL.

Kuncak, V. and M. Rinard: 2005, 'Decision Procedures for Set-Valued Fields'. In: *1st International Workshop on Abstract Interpretation of Object-Oriented Languages (AIOOL 2005)*.

Kuncak, V. and M. Rinard: 2006, 'An overview of the Jahob analysis system: Project Goals and Current Status'. In: *NSF Next Generation Software Workshop*.

Lahiri, S. K. and S. A. Seshia: 2004, 'The UCLID Decision Procedure'. In: *CAV'04*.

Lam, P., V. Kuncak, and M. Rinard: 2004, 'Generalized Typestate Checking Using Set Interfaces and Pluggable Analyses'. *SIGPLAN Notices* **39**, 46–55.

Lam, P., V. Kuncak, and M. Rinard: 2005, 'Generalized Typestate Checking for Data Structure Consistency'. In: *6th International Conference on Verification, Model Checking and Abstract Interpretation*.

Loewenheim, L.: 1915, 'Über Mögligkeiten im Relativkalkül'. *Math. Annalen* **76**, 228–251.

Marnette, B., V. Kuncak, and M. Rinard: 2005, 'On Algorithms and Complexity for Sets with Cardinality Constraints'. Technical report, MIT CSAIL.

Marriott, K. and M. Odersky: 1994, 'Negative Boolean Constraints'. Technical Report 94/203, Monash University.

Martin, U. and T. Nipkow: 1989, 'Boolean Unification: The Story So Far'. *Journal of Symbolic Computation* **7**(3), 275–293.

Møller, A. and M. I. Schwartzbach: 2001, 'The Pointer Assertion Logic Engine'. In: *Programming Language Design and Implementation*.

Nelson, G. and D. C. Oppen: 1979, 'Simplification by Cooperating Decision Procedures'. *ACM TOPLAS* **1**(2), 245–257.

Nipkow, T., L. C. Paulson, and M. Wenzel: 2002, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, Vol. 2283 of *LNCS*. Springer-Verlag.

Ohlbach, H. J. and J. Koehler: 1998, 'How to Extend a Formal System with a Boolean Algebra Component'. In: W. B. P.H. Schmidt (ed.): *Automated Deduction. A Basis for Applications*, Vol. III. Kluwer Academic Publishers, pp. 57–75.

Owre, S., J. M. Rushby, and N. Shankar: 1992, 'PVS: A Prototype Verification System'. In: D. Kapur (ed.): *11th CADE*, Vol. 607 of *LNAI*. pp. 748–752.

Papadimitriou, C. H.: 1981, 'On the Complexity of Integer Programming'. *J. ACM* **28**(4), 765–768.

Podelski, A. and A. Rybalchenko: 2005, 'Transition Predicate Abstraction and Fair Termination'. In: *ACM POPL*.

Presburger, M.: 1929, '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*. pp. 92–101.

Pugh, W.: 1991, 'The Omega test: a fast and practical integer programming algorithm for dependence analysis'. In: *Supercomputing '91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing*. pp. 4–13.

Reddy, C. R. and D. W. Loveland: 1978, 'Presburger Arithmetic with Bounded Quantifier Alternation'. In: *ACM STOC*. pp. 320–325.

Revesz, P.: 2004, 'Quantifier-Elimination for the First-Order Theory of Boolean Algebras with Linear Cardinality Constraints'. In: *Proc. Advances in Databases and Information Systems (ADBIS'04)*, Vol. 3255 of *LNCS*.

Ruess, H. and N. Shankar: 2001, 'Deconstructing Shostak'. In: *Proc. 16th IEEE LICS*.

Rugina, R.: 2004, 'Quantitative Shape Analysis'. In: *Static Analysis Symposium (SAS'04)*.

Sagiv, M., T. Reps, and R. Wilhelm: 2002, 'Parametric shape analysis via 3-valued logic'. *ACM TOPLAS* **24**(3), 217–298.

Skolem, T.: 1919, '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.

Thatcher, J. W. and J. B. Wright: 1968, 'Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic'. *Mathematical Systems Theory* **2**(1), 57–81.

Thomas, W.: 1997, 'Languages, Automata, and Logic'. In: *Handbook of Formal Languages Vol.3: Beyond Words*. Springer-Verlag.

Tinelli, C. and C. Zarba: 2005, 'Combining Nonstably Infinite Theories'. *Journal of Automated Reasoning* **34**(3).

Tiwari, A.: 2000, 'Decision procedures in automated deduction'. Ph.D. thesis, Department of Computer Science, State University of New York at Stony Brook.

Voronkov, A.: 1995, 'The anatomy of Vampire (implementing bottom-up procedures with code trees)'. *Journal of Automated Reasoning* **15**(2), 237–265.

Weidenbach, C.: 2001, 'Combining Superposition, Sorts and Splitting'. In: A. Robinson and A. Voronkov (eds.): *Handbook of Automated Reasoning*, Vol. II. Elsevier Science, Chapt. 27, pp. 1965–2013.

Yorsh, G., T. Reps, and M. Sagiv: 2004, 'Symbolically Computing Most-Precise Abstract Operations for Shape Analysis'. In: *10th TACAS*.

Zarba, C. G.: 2004a, 'The Combination Problem in Automated Reasoning'. Ph.D. thesis, Stanford University.

Zarba, C. G.: 2004b, 'Combining Sets with Elements'. In: N. Dershowitz (ed.): *Verification: Theory and Practice*, Vol. 2772 of *Lecture Notes in Computer Science*. pp. 762–782.

Zarba, C. G.: 2004c, 'A Quantifier Elimination Algorithm for a Fragment of Set Theory Involving the Cardinality Operator'. In: *18th International Workshop on Unification*.

Zarba, C. G.: 2005, 'Combining Sets with Cardinals'. *Journal of Automated Reasoning* **34**(1).

Zhang, L. and S. Malik: 2002, 'Conflict driven learning in a quantified Boolean Satisfiability solver'. In: *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*. New York, NY, USA, pp. 442–449.