An Algorithm for Deciding BAPA: Boolean Algebra with Presburger Arithmetic

Viktor Kuncak [1], Huu Hai Nguyen [2], and Martin Rinard [1,2]

[1] MIT CSAIL, 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 sets, and supports arbitrary quantification over both sets and integers.

Our 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 annotations contain quantifiers themselves, or when proving simulation relation conditions for refinement and equivalence of program fragments. Furthermore, BAPA constraints can be used to extend the techniques for proving the termination of integer programs to programs that manipulate data structures, and have applications in constraint databases.

We give a formal description of a decision procedure for BAPA, which implies the decidability of the satisfiability and validity problems for BAPA. We analyze our algorithm and obtain an elementary upper bound on the running time, thereby giving the first complexity bound for BAPA. 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. Our algorithm can also be used to yield a space-optimal decision procedure for BA though a reduction to PA with bounded quantifiers.

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 with the algorithm is promising.

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 [22, 28] 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 realizable counterexample or with a sound claim 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 [41–43, 80] we identify the first-order theory of Boolean algebras (BA) as a

Compiled March 11, 2005, 6:59pm for submission to CADE-20.

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. BA is known to be decidable [45, 67].

The motivation for this paper is the fact that we often need to reason not only about the data structure content, but also about the size of the 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 [19] (which also proves the well-known Feferman-Vaught theorem [29, Section 9.6] about the products of first-order theories). Recently, a decision procedure for a fragment of BAPA without quantification over sets was presented in [79], cast as a multi-sorted theory. Starting from [43] as our motivation, we have observed in [38] the decidability of the full BAPA (which was initially left open in [79]). After our report [38], an algorithm for a language between BA and BAPA was presented in [62] as a way of evaluating queries in constraint databases. The constraints in [62] allow only constant integer parameters and not integer variables; moreover, [62] still leaves open the complexity of the algorithm.

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 an elementary upper bound on the complexity of BAPA. Our result is the first upper complexity bound on BAPA; along with a lower bound from PA, we obtain a good estimate of BAPA worst-case complexity. We have also implemented our decision procedure; we report on our initial experience in using the decision procedure in the context of a system for checking data structure consistency.

Contributions. We summarize the contributions of our paper as follows.

1. As a **motivation** for BAPA, we show in Section 3 how BAPA constraints can be used for program analysis and verification by expressing 1) data structure invariants, 2) the correctness of procedures with respect to their specifications, 3) simulation relations between program fragments, and 4) termination conditions for programs that manipulate data structures.
2. We present an **algorithm** α (Section 4) that translates BAPA sentences into PA sentences by translating set quantifiers into integer quantifiers. The algorithm is surprisingly simple (the entire source code is included in the Appendix, Section 12) and shows a deep connection between BA and PA.
3. We analyze our algorithm α and show that it yields an **elementary upper bound** on the worst-case complexity of the validity problem for BAPA sentences that is close

to the bound on PA sentences themselves (Section 5). This is the first complexity bound for BAPA, and is the main contribution of this paper.
4. We discuss our experience in using our **implementation** of BAPA to discharge verification conditions generated in the Jahob verification system [34].
5. In addition, we note the following related complexity, decidability and undecidability results:
   (a) We show that PA sentences generated by translating pure BA sentences can be checked for validity in singly exponential space, which is a good bound in the light of alternating exponential lower bound for BA (Section 5.2).
   (b) We show how to extend our algorithm to **infinite sets** and predicates for distinguishing finite and infinite sets (Section 10).
   (c) We examine the relationship of our results to the monadic second-order logic (MSOL) of strings (Section 11). In contrast to the undecidability of MSOL with equicardinality operator (Section 11.2), we identify a combination of MSOL over trees with BA that is **decidable**. This result follows from the fact that our algorithm α enables adding BA operations to any extension of PA, including decidable extensions such as MSOL over strings (Section 11.1).

A preliminary version of our results, including the algorithm and complexity analysis appear in [38], which also contains some background on quantifier elimination.

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 [40]. Our language for BA (Figure 1) allows cardinality constraints of the form $|A| = C$ where C is a *constant* integer. Such constant cardinality constraints are useful and enable quantifier elimination for the resulting language [45,67]. However, they do not allow stating constraints such as $|A| = |B|$ for two sets A and B, and cannot represent constraints on changing program variables. Consider therefore the equicardinality relation $\mathsf{eqcard}(A, B)$ that holds iff $|A| = |B|$, and consider BA extended with relation $\mathsf{eqcard}(A, B)$. Define the ternary relation $\mathsf{plus}(A, B, C) \iff (|A| = |B| + |C|)$ by the formula $\exists x_1. \exists x_2. x_1 \cap x_2 = \emptyset \wedge C = x_1 \cup x_2 \wedge \mathsf{eqcard}(A, x_1) \wedge \mathsf{eqcard}(B, x_2)$. The relation $\mathsf{plus}(A, B, C)$ allows us to express addition using arbitrary sets as representatives for natural numbers. Moreover, we can represent integers as equivalence classes of pairs of natural numbers under the equivalence relation $(x, y) \sim (u, v) \iff x + v = u + y$. This construction 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 [19, Section 8], which showed the decidability of BA over sets extended with the equicardinality relation eqcard, using the decidability of the first-order theory of the addition of cardinal numbers.

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

Fig. 1. Formulas of Boolean Algebra (BA)

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

Fig. 2. Formulas of Presburger Arithmetic (PA)

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

Fig. 3. Formulas of Boolean Algebras 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: 1) subsumes the language of PA in Figure 2; 2) subsumes the language of BA in Figure 3; and 3) 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 operations in standard way, and interpret sets in boolean algebra over subsets of a finite sets. The MAXC constant denotes the size of the finite universe $\mathcal{U}$, so we require $\mathsf{MAXC} = |\mathcal{U}|$ in all models. (Our results also extend to infinite sets, see Section 10 for the discussion.)

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. Finally, section 3.3 shows how BAPA can be used to express and prove termination conditions for a class of programs.

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 [18, 66] or can be derived from executable programs using an abstraction process [41, 43]. 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 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.

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

Fig. 4. An Example Procedure

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

Fig. 5. Hoare Triple for insert Procedure

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

Fig. 6. Verification Condition for Figure 5

In addition to the explicit requires and ensures clauses, 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 clause of the procedure. The Hoare triple [28] 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 particularly simple formula; in general, formulas that arise in the compositional analysis of set programs with integer variables may contain alternations of existential and universal variables over both integers and sets. This paper shows the decidability of such formulas and presents the complexity of the decision procedure.

3.2 Proving Simulation Relation Conditions

Another example of where BAPA constraints are useful is when proving that a given 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 into the set of running processes, 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 of all processes and a number 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 [46] is $\forall s_1.\forall s_1'.\forall s_2.(t_1(s_1,s_1') \land r(s_1,s_2)) \Rightarrow \exists s_2'.\,(t_2(s_2,s_2') \land r(s_2,s_2'))$. 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. $\forall s_1.\forall s_1'.\forall s_2.\exists s_2'.\; r(s_1,s_2) \land \mathsf{post}_1(s_1,s_1') \land \mathsf{pre}_2(s_2) \Rightarrow \mathsf{post}_2(s_2,s_2') \land r(s_2,s_2')$

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.

var R : **set**;
var S : **set**;

procedure start1(x)
requires $x \not\subseteq \mathsf{R} \land |x| = 1 \land |R| < \mathsf{MAXR}$
ensures $\mathsf{R}' = \mathsf{R} \cup x \land \mathsf{S}' = \mathsf{S}$
{
  $\mathsf{R} := \mathsf{R} \cup x$;
}

var P : **set**;
var k : **set**;

procedure start2(x)
requires $x \not\subseteq \mathsf{P} \land |x| = 1 \land k < \mathsf{MAXR}$
ensures $\mathsf{P}' = \mathsf{P} \cup x \land \mathsf{k}' = \mathsf{k} + 1$
{
  $\mathsf{P} := \mathsf{P} \cup x$;
  $\mathsf{k} := \mathsf{k} + 1$;
}

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

Simulation relation conditions in BAPA:

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

Fig. 7. Proving simulation relation in BAPA

3.3 Proving Termination of Programs

We next show how 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

```
var iter : set;

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

Fig. 8. Terminating program

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:
$$\forall \mathsf{iter}. \forall \mathsf{iter}'.\ (\exists e. |e| = 1 \land e \subseteq \mathsf{iter} \land \mathsf{iter}' = \mathsf{iter} \setminus e) \Rightarrow |\mathsf{iter}'| < |\mathsf{iter}|$$

Fig. 9. Termination proof for Figure 8

program state into a non-negative integer, and the 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 end of the procedure, 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. Note that, because the loop contains a local variable, the resulting loop transition relation contains an existential quantifier. The resulting termination condition can be expressed as a formula that 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 fragments of a terminating program that manipulates only integers, which can be proved terminating using techniques [55–57]. The simulation relation condition can be proved correct using our BAPA decision procedure whenever the simulation relation is expressible with a BAPA formula.

4 Decision Procedure for BAPA

This section presents our algorithm, denoted α, which 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 the space and time bounds for PA sentences [61], the algorithm α yields reasonable space and time bounds for deciding BAPA sentences (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 [23, 31, 44].
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 See Section 11 for the discussion.

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. Because of this feature, we can use the algorithm α to project out local state components from formulas that describe invariants and transition relations, and simplify the resulting formulas.

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

$$Q_p v_p. \ldots Q_1 v_1.\ F(v_1, \ldots, v_p) \tag{1}$$

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 BA part and PA part. To achieve this, replace each formula $x = y$ where x and y are sets, with the conjunction $x \subseteq y \wedge y \subseteq x$, and replace each formula $x \subseteq y$ with the equivalent formula $|x \cap y^c| = 0$. In the resulting formula, each set 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 Venn diagram [74]) of the form $\bigwedge_{i=1}^{n} x_i^{\alpha_i}$ where $x_i^{\alpha_i}$ is either x_i or x_i^c; hence there are $m = 2^n$ cubes $s_1, \ldots, s_m$. Suppose that $t(x) = s_{j_1} \cup \ldots s_{j_a}$; 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. Then the resulting formula is equivalent to

$$\begin{array}{l} Q_p v_p. \ldots Q_1 v_1. \\ \qquad \exists^+ l_1, \ldots, l_m.\ \bigwedge_{i=1}^{m} |s_i| = l_i \ \wedge\ G_1 \end{array} \tag{2}$$

where G_1 is a PA formula and $m = 2^n$. Formula (2) 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

$$\begin{array}{l} Q_p v_p \ldots Q_r v_r. \\ \qquad \exists^+ l_1 \ldots l_q.\ \bigwedge_{i=1}^{q} |s_i| = l_i \ \wedge\ G_r \end{array} \tag{3}$$

where G_r is a PA formula, r grows from 1 to $p+1$, and $q = 2^e$ where e for $0 \le e \le n$ 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 (3). 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 $Q_r v_r$.

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.

For universal quantifiers, 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 the existential quantifier is used as a let-binding, 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. Given that the universal quantifier $\forall k$ can be represented as a sequence of unary operators $\neg\exists k \neg$, from the elimination of $\exists k$ we immediately obtain the elimination of $\forall k$; it turns out that it suffices to let $G_{r+1} = \forall k. G_r$.

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

$$\exists y.\ \exists^{+} l_1 \ldots l_q.\ \bigwedge_{i=1}^{q} |s_i| = l_i \ \wedge\ G_r \tag{4}$$

which is equivalent to $\exists^{+} l_1 \ldots l_q.\ (\exists y. \bigwedge_{i=1}^{q} |s_i| = l_i) \ \wedge\ G_r$. This is the key step of the algorithm and relies on the following lemma, whose proof is in Section 9.

Lemma 1. *Let $b_1, \ldots, b_n$ be finite disjoint sets, and $l_1, \ldots, l_n, k_1, \ldots, k_n$ be natural numbers. Then the following two statements are equivalent: (1) There exists a finite set y such that $\bigwedge_{i=1}^{n} |b_i \cap y| = k_i \wedge |b_i \cap y^c| = l_i$ and (2) $\bigwedge_{i=1}^{n} |b_i| = k_i + l_i$. Moreover, the statement continues to hold if for any subset of indices i the conjunct $|b_i \cap y| = k_i$ is replaced by $|b_i \cap y| \geq k_i$ or the conjunct $|b_i \cap y^c| = l_i$ is replaced by $|b_i \cap y^c| \geq l_i$, provided that $|b_i| = k_i + l_i$ is replaced by $|b_i| \geq k_i + l_i$, as indicated in Figure 10.*

original formula	eliminated form
$\exists y.\ \ldots\ |b \cap y| \geq k \wedge |b \cap y^c| \geq l\ \ldots$	$|b| \geq k + l$
$\exists y.\ \ldots\ |b \cap y| = k \wedge |b \cap y^c| \geq l\ \ldots$	$|b| \geq k + l$
$\exists y.\ \ldots\ |b \cap y| \geq k \wedge |b \cap y^c| = l\ \ldots$	$|b| \geq k + l$
$\exists y.\ \ldots\ |b \cap y| = k \wedge |b \cap y^c| = l\ \ldots$	$|b| = k + l$

Fig. 10. Rules for Eliminating Quantifiers from Boolean Algebra Expressions

In the quantifier elimination step, assume without loss of generality that the set variables $s_1, \ldots, s_q$ 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 formula

$$\exists^{+} l_1 \ldots l_q.\ \bigwedge_{i=1}^{q'} |s'_i| = l_{2i-1} + l_{2i} \ \wedge\ G_r \tag{5}$$

for $q' = 2^{e-1}$. Finally, to obtain a formula of the form (3) for $r + 1$, introduce fresh variables l'_i constrained by $l'_i = l_{2i-1} + l_{2i}$, rewrite (5) 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

$$G_{r+1} \equiv \exists^+ l_1 \dots l_q. \bigwedge_{i=1}^{q'} l'_i = l_{2i-1} + l_{2i} \ \wedge \ G_r \tag{6}$$

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

To eliminate a set quantifier $\forall y$, proceed analogously: introduce fresh variables $l'_i = l_{2i-1} + l_{2i}$ and let $G_{r+1} \equiv \forall^+ l_1 \dots l_q. \ (\bigwedge_{i=1}^{q'} l'_i = l_{2i-1} + l_{2i}) \ \Rightarrow \ G_r$, which can be verified by expressing $\forall y$ as $\neg \exists y \neg$.

After eliminating all quantifiers as described above, we obtain a formula of the form $\exists^+ l. \ |\mathcal{U}| = l \wedge G_{p+1}(l)$. We define the result of the algorithm, denoted $\alpha(F_0)$, to be the PA sentence $G_{p+1}(\mathsf{MAXC})$.

This completes the description of the algorithm α. Given that the validity of PA sentences is decidable, the algorithm α is a decision procedure for BAPA sentences.

Theorem 2. *The algorithm α described above maps each* BAPA*-sentence F_0 into an equivalent* PA*-sentence $\alpha(F_0)$.*

Formalization of the algorithm α. To formalize the algorithm α, we have implemented it the functional programming language O'Caml; Section 12 contains the source code of the implementation. As an illustration, when we run the implementation on the BAPA formula in Figure 6 which represents a verification condition, we immediately obtain the PA formula in Figure 11. 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 12 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 12.

The subsequent sections explore the consequences of the existence of the algorithm α, including an upper bound on the computational complexity of BAPA sentences and the combination of BA with proper extensions of PA. We explain our experience with using the implementation in Section 6.

5 Complexity

In this section we analyze the algorithm α from Section 4 and obtain space and time bounds on BAPA from the corresponding space and time bounds for PA. We then show that the new decision procedure meets good worst-case space bounds for BA if applied to BA formulas. Moreover, by construction, our procedure reduces to the procedure for Presburger arithmetic formulas if there are no set quantifiers. In summary, our decision procedure is reasonable for BA, does not impose any overhead for pure PA formulas, and the complexity of the general BAPA validity has the same height of the tower of exponentials as the complexity of PA itself.

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

Fig. 11. The translation of the BAPA sentence from Figure 6 into a PA sentence

general relationship:

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

in this example:

$$
\begin{array}{l}
\mathsf{set}_1 = \mathsf{content}' \\
\mathsf{set}_2 = \mathsf{content} \\
\mathsf{set}_3 = e \\
l_{000} = |\mathsf{content}'^c \cap \mathsf{content}^c \cap e^c| \\
l_{001} = |\mathsf{content}'^c \cap \mathsf{content}^c \cap e| \\
l_{010} = |\mathsf{content}'^c \cap \mathsf{content} \cap e^c| \\
l_{011} = |\mathsf{content}'^c \cap \mathsf{content} \cap e| \\
l_{100} = |\mathsf{content}' \cap \mathsf{content}^c \cap e^c| \\
l_{101} = |\mathsf{content}' \cap \mathsf{content}^c \cap e| \\
l_{110} = |\mathsf{content}' \cap \mathsf{content} \cap e^c| \\
l_{111} = |\mathsf{content}' \cap \mathsf{content} \cap e|
\end{array}
$$

Fig. 12. The Correspondence between Integer Variables in Figure 11 and Set Variables in Figure 6

5.1 An Elementary Upper Bound

We next show that the algorithm in Section 4 transforms a BAPA sentence F_0 into a PA sentence whose size is at most one exponential larger and which has the same number of quantifier alternations.

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. Define the iterated exponentiation function $\mathsf{exp}_k(x)$ by $\mathsf{exp}_0(x) = x$ and $\mathsf{exp}_{k+1}(x) = 2^{\mathsf{exp}_k(x)}$. We have the following lemma.

Lemma 3. *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 $2^{O(\mathsf{size}(F_0))}$ space.*

We next consider the worst-case space bound on BAPA. Recall first the following bound on space complexity for PA.

Fact 1 *[20, Chapter 3] The validity of a* PA *sentence of length n can be decided in space* $\mathsf{exp}_2(O(n))$.

From Lemma 3 and Fact 1 we conclude that the validity of BAPA formulas can be decided in space $\mathsf{exp}_3(O(n))$. It turns out, however, that we obtain better bounds on BAPA validity by analyzing the number of quantifier alternations in BA and BAPA formulas.

Fact 2 *[61] The validity of a* PA *sentence of length n and the number of quantifier alternations m can be decided in space $2^{n^{O(m)}}$.*

From Lemma 3 and Fact 2 we obtain our space upper bound, which implies the upper bound on deterministic time.

Theorem 4. *The validity of a* BAPA *sentence of length n and the number of quantifier alternations m can be decided in space $\mathsf{exp}_2(O(mn))$, and, consequently, in deterministic time $\mathsf{exp}_3(O(mn))$.*

If we approximate quantifier alternations by formula size, we conclude that BAPA validity can be decided in space $\mathsf{exp}_2(O(n^2))$ compared to $\mathsf{exp}_2(O(n))$ bound for Presburger arithmetic from Fact 1. Therefore, despite the exponential explosion in the size of the formula in the algorithm α, thanks to the same number of quantifier alternations, our bound is not very far from the bound for Presburger arithmetic.

5.2 BA as a Special Case

We next analyze the result of applying the algorithm α to a BA sentence F_0. By a 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 best upper bounds for PA are worse than the corresponding bounds for BA. However, we identify a special form of PA sentences $\mathsf{PA}_{\mathsf{BA}} = \{\alpha(F_0) \mid F_0 \text{ is in } \mathsf{BA}\}$ and show that such sentences can be decided in $2^{O(n)}$ space, which is good for BA [32]. Our analysis shows that using binary representations of integers that correspond to the sizes of sets achieves a similar effect to representing these sets as bitvectors, although the two representations are not identical.

Let S be the number of set variables in the initial formula F_0 (recall that set variables are the only variables in F_0). Let $l_1, \ldots, l_q$ be the set of free variables of the formula $G_r(l_1, \ldots, l_q)$; 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 5. *For each r where $1 \leq r \leq S$ the truth value of $G_r(w_1, \ldots, w_q)$ is equal to the the truth value of $G_r(\bar{w}_1, \ldots, \bar{w}_q)$ where $\bar{w}_i = \min(w_i, 2^{r-1})$.*

Now consider a formula F_0 of size n with S free variables. Then $\alpha(F_0) = G_{S+1}$. By Lemma 3, $\mathsf{size}(\alpha(F_0))$ is $O(nS2^S)$. By Lemma 5, it suffices for the outermost variable k 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 $2^{S+1} - 1$ variables can be represented in $O(S)$ space, which is the same order of space used to represent the names of variables themselves. This means that evaluating the formula $\alpha(F_0)$ can be done in the same space $O(nS2^S)$ as the size of the formula. Representing the valuation assigning values to variables can be done in $O(S2^S)$ space, so the truth value of the formula can be evaluated in $O(nS2^S)$ space, which is certainly $2^{O(n)}$. We obtain the following theorem.

Theorem 6. *If F_0 is a pure* BA *formula with S variables and of size n, then the truth value of $\alpha(B_0)$ can be computed in $O(nS2^S)$ and therefore $2^{O(n)}$ space.*

6 Experience Using Our Decision Procedure for BAPA

We have experimented with BAPA in the context of Jahob system [34] 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 currently contains interfaces to the Isabelle interactive theorem prover [51], the Simplify theorem prover [17] as well as the Omega Calculator [60] and the LASH [44] decision procedures for PA.

Using Jahob, we have generated verification conditions for several Java program fragments that require reasoning about sets and their cardinalities, for example proving the equality relation between the number of elements in a list and the integer field size after they have been updated. Formulas arising from examples in Section 3 have also been discharged using our current implementation. We have found that Simplify is able to deal with some of the formulas involving only sets or only integers, but not with formulas that relate cardinalities of operations on sets to 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 current implementation makes use of some transformations and simplifications to reduce formula sizes. We find that eliminating set variables early by substitution is a highly effective optimization. When using Omega Calculator as the backend for our system, we also observed that lifting quantifiers to the top level noticeably improve performance. These transformations effectively extend the range of formulas that the current system can handle. Our current implementation of the decision procedure and example formulas can be found on the website [33].

7 Related Work

Our paper is the first result that shows a complexity bound for the first-order theory of BAPA. The decidability for BAPA, presented as BA with equicardinality constraints was presented in [19] (see Section 2). A decision procedure for a special case of BAPA was presented in [79], which allows only quantification over *elements* but not over *sets* of elements. BAPA is a more general language because singleton sets can represent elements, so quantification over sets allows modelling quantification over elements. [62] (which appeared after [38]) shows the decidability of BA with constant cardinalities.

Presburger arithmetic. The original result on decidability of PA is [59]. The best known bound on formula size is [20]. This decision procedure was improved in [16] and subsequently in [52]. An analysis based on the number of quantifier alternations is presented in [61]. Our implementation uses quantifer-elimination based Omega test [60] which, in our current experience, outperforms other implementations we have tried. Among the decision procedures for full PA, [13] is the only proof-generating version, and is based on a version of [16]. Decidable fragments of arithmetic that go beyond PA include MSOL over strings [11, 31] and [9].

Boolean Algebras. The first results on decidability of BA are from [45], [1, Chapter 4] and use quantifier elimination, from which one can derive small model property; [32] gives the complexity of the satisfiability problem. [48] studies unification in Boolean rings. The quantifier-free fragment of BA is shown NP-complete in [47]; see [39] for a generalization of this result using parameterized complexity of the Bernays-Schönfinkel-Ramsey class of first-order logic [8, Page 258] which can be decided using [24] or [7]. [12] 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. Quantifier-free formulas are also used in constraint solving [2, 6, 15]. Among the systems for interactively reasoning about richer theories of sets are Isabelle [51], HOL [26], PVS [53], TPS [3]; first-order frameworks such as Athena [4] can use axiomatizations of sets along with calls to resolution-based theorem provers such as Vampire [75] to reason about sets.

Combinations of Decidable Theories. The techniques for combining *quantifier-free* theories [50,63] and their generalizations such as [71–73,77,78] are of great importance for program verification. Our paper shows a particular combination result for *quantified formulas*, which add additional expressive power in writing specifications. Among the general results for quantified formulas are the Feferman-Vaught theorem for products [19] and term powers [36, 37]. While we have found quantifiers to be useful in several contexts, many problems can be encoded in quantifier-free formulas, so it is interesting to consider a combination of BAPA with solvers for quantifier-free formulas [21,25,69]. Description logics [5] and two-variable logic with counting [27,54,58] support sets and cardinalities, and additionally support relations, but do not allow quantification over sets.

Analyses of Dynamic 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 [35, 49, 65, 76]. Recently, these approaches were extended to handle the combinations of the constraints representing data structure contents and constraints representing numerical properties of data structures [14, 64]. 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. As we have illustrated in Section 3, the use of quantifiers is important for proving verification conditions that include quantified annotations, for computing abstractions of program fragments that involve local variables, and for proving simulation relation conditions. We has also illustrated the use of BAPA for reasoning about termination of programs that manipulate dynamic data structures by associating integer variables with sizes of sets that specify the objects in data structures and using techniques for proving termination of programs with integers [55–57]. Other possible applications of our decision procedure include query evaluation in constraint databases [62] and loop invariant inference [30].

8 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 algebras with Presburger arithmetic (BAPA), showed an elementary upper bound on the worst-case complexity, implemented the algorithm and applied it to several reasoning tasks. Our experience indicates that the algorithm will be useful as a component of a decision procedure of our data structure verification system.

Acknowledgements. We thank Alexis Bes for pointing out the relevance of [19, Section 8], Chin Wei-Ngan for useful discussions on the analysis of data structure size constraints and useful comments on a version of this paper, Calogero Zarba for comments on an earlier version of this paper, Peter Revesz for pointing to his recent paper [62], Andreas Podelski for discussions about transition relations, Bruno Courcelle on remarks regarding undecidability of MSOL with equicardinality constraints, Cesare Tinelli and Konstantin Korovin on discussions of Bernays-Schönfinkel-Ramsey class, the members of the Stanford REACT group and the Berkeley CHESS group on useful discussions on decision procedures and program analysis.

References

1. W. Ackermann. *Solvable Cases of the Decision Problem*. North Holland, 1954.
2. Alex Aiken, Dexter Kozen, Moshe Vardi, and Ed Wimmers. The complexity of set constraints. In *Proceedings of Computer Science Logic 1993*, pages 1–17, September 1993.
3. Peter B. Andrews, Sunil Issar, Dan Nesmith, and Frank Pfenning. The tps theorem proving system. In *10th CADE*, volume 449 of *LNAI*, pages 641–642, 1990.
4. Konstantine Arkoudas, Karen Zee, Viktor Kuncak, and Martin Rinard. Verifying a file system implementation. In *Sixth International Conference on Formal Engineering Methods (ICFEM'04)*, volume 3308 of *LNCS*, Seattle, Nov 8-12, 2004 2004.
5. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
6. Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Set constraints are the monadic class. In *Logic in Computer Science*, pages 75–83, 1993.
7. Peter Baumgartner and Cesare Tinelli. The Model Evolution Calculus. In Franz Baader, editor, *CADE-19 – The 19th International Conference on Automated Deduction*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 350–364. Springer, 2003.
8. Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.
9. M. Bozga and R. Iosif. On decidability within the arithmetic of addition and divisibility. In *FOSSACS'05*, 2005.
10. V. Bruyére, G. Hansel, C. Michaux, and R. Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc. Simon Stevin*, 1:191–238, 1994.
11. J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.
12. Domenico Cantone, Eugenio Omodeo, and Alberto Policriti. *Set Theory for Computing*. Springer, 2001.
13. Amine Chaieb and Tobias Nipkow. Generic proof synthesis for presburger arithmetic. Technical report, Technische Universität München, October 2003.
14. Wei-Ngan Chin, Siau-Cheng Khoo, and Dana N. Xu. Extending sized types with with collection analysis. In *ACM SIGPLAN Workshop on Partial Evaluation and Semantics Based Program Manipulation (PEPM'03)*, 2003.
15. Brahim Hnich Christian Bessiere, Emmanuel Hebrard and Toby Walsh. Disjoint, partition and intersection constraints for set and multiset variables. In *CP'04*, pages 138–152, 2004.
16. D. C. Cooper. Theorem proving in arithmetic without multiplication. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 7, pages 91–100. Edinburgh University Press, 1972.
17. David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Laboratories Palo Alto, 2003.

18. Robert K. Dewar. Programming by refinement, as exemplified by the SETL representation sublanguage. *Transactions on Programming Languages and Systems*, July 1979.
19. S. Feferman and R. L. Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.
20. Jeanne Ferrante and Charles W. Rackoff. *The Computational Complexity of Logical Theories*, volume 718 of *Lecture Notes in Mathematics*. Springer-Verlag, 1979.
21. Cormac Flanagan, Rajeev Joshi, Xinming Ou, and James B. Saxe. Theorem proving using lazy proof explication. In *CAV*, pages 355–367, 2003.
22. Robert W. Floyd. Assigning meanings to programs. In *Proc. Amer. Math. Soc. Symposia in Applied Mathematics*, volume 19, pages 19–31, 1967.
23. Vijay Ganesh, Sergey Berezin, and David L. Dill. Deciding presburger arithmetic by model checking and comparisons with other methods. In *Formal Methods in Computer-Aided Design*. Springer-Verlag, November 2002.
24. H. Ganzinger and K. Korovin. Integrating equational reasoning into instantiation-based theorem proving. In *Computer Science Logic (CSL'04)*, volume 3210 of *Lecture Notes in Computer Science*, pages 71–84. Springer, 2004.
25. Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification, CAV'04 (Boston, Massachusetts)*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004.
26. M. J. C. Gordon and T. F. Melham. *Introduction to HOL, a theorem proving environment for higher-order logic*. Cambridge University Press, Cambridge, England, 1993.
27. Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science LICS '97, Warschau*, 1997.
28. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
29. Wilfrid Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1993.
30. Deepak Kapur. Automatically generating loop invariants using quantifier elimination. In *IMACS Intl. Conf. on Applications of Computer Algebra*, 2004.
31. Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. MONA implementation secrets. In *Proc. 5th International Conference on Implementation and Application of Automata*. LNCS, 2000.
32. Dexter Kozen. Complexity of boolean algebras. *Theoretical Computer Science*, 10:221–247, 1980.
33. Viktor Kuncak. BAPA web page. http://www.mit.edu/~vkuncak/projects/bapa/, 2004.
34. Viktor Kuncak. The Jahob project web page. http://www.mit.edu/~vkuncak/projects/jahob/, 2004.
35. Viktor Kuncak, Patrick Lam, and Martin Rinard. Role analysis. In *Proc. 29th POPL*, 2002.
36. Viktor Kuncak and Martin Rinard. On the theory of structural subtyping. Technical Report 879, Laboratory for Computer Science, Massachusetts Institute of Technology, 2003.
37. Viktor Kuncak and Martin Rinard. Structural subtyping of non-recursive types is decidable. In *Eighteenth Annual IEEE Symposium on Logic in Computer Science*, 2003.
38. Viktor Kuncak and Martin Rinard. The first-order theory of sets with cardinality constraints is decidable. Technical Report 958, MIT CSAIL, July 2004.
39. Viktor Kuncak and Martin Rinard. Decision procedures for set-valued fields. In *1st International Workshop on Abstract Interpretation of Object-Oriented Languages (AIOOL 2005)*, 2005.
40. Patrick Lam, Viktor Kuncak, and Martin Rinard. Generalized typestate checking using set interfaces and pluggable analyses. *SIGPLAN Notices*, 39:46–55, March 2004.

41. Patrick Lam, Viktor Kuncak, and Martin Rinard. On our experience with modular pluggable analyses. Technical Report 965, MIT CSAIL, September 2004.
42. Patrick Lam, Viktor Kuncak, and Martin Rinard. Cross-cutting techniques in program specification and analysis. In *4th International Conference on Aspect-Oriented Software Development (AOSD 2005)*, 2005.
43. Patrick Lam, Viktor Kuncak, and Martin Rinard. Generalized typestate checking for data structure consistency. In *6th International Conference on Verification, Model Checking and Abstract Interpretation*, 2005.
44. LASH. The LASH Toolset. `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.
45. L. Loewenheim. Über mögligkeiten im relativkalkül. *Math. Annalen*, 76:228–251, 1915.
46. Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part I: Untimed systems. *Information and Computation, 121(2)*, 1995.
47. Kim Marriott and Martin Odersky. Negative boolean constraints. Technical Report 94/203, Monash University, August 1994.
48. Ursula Martin and Tobias Nipkow. Boolean unification: The story so far. *Journal of Symbolic Computation*, 7(3):275–293, 1989.
49. Anders Møller and Michael I. Schwartzbach. The Pointer Assertion Logic Engine. In *Proc. ACM PLDI*, 2001.
50. Greg Nelson. Techniques for program verification. Technical report, XEROX Palo Alto Research Center, 1981.
51. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.
52. Derek C. Oppen. Elementary bounds for presburger arithmetic. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 34–37. ACM Press, 1973.
53. S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th CADE*, volume 607 of *LNAI*, pages 748–752, jun 1992.
54. 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.
55. Andreas Podelski and Andrey Rybalchenko. A complete method for synthesis of linear ranking functions. In *VMCAI'04*, 2004.
56. Andreas Podelski and Andrey Rybalchenko. Transition invariants. In *LICS'04*, 2004.
57. Andreas Podelski and Andrey Rybalchenko. Transition predicate abstraction and fair termination. In *POPL'05*, 2005.
58. Ian Pratt-Hartmann. Complexity of the two-variable fragment with (binary-coded) counting quantifiers. *CoRR*, cs.LO/0411031, 2004.
59. M. 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.
60. William Pugh. 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*, pages 4–13. ACM Press, 1991.
61. C. R. Reddy and D. W. Loveland. Presburger arithmetic with bounded quantifier alternation. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 320–325. ACM Press, 1978.
62. Peter Revesz. Quantifier-elimination for the first-order theory of boolean algebras with linear cardinality constraints. In *Proc. Advances in Databases and Information Systems (ADBIS'04)*, volume 3255 of *LNCS*, 2004.
63. Harald Ruess and Natarajan Shankar. Deconstructing Shostak. In *Proc. 16th IEEE LICS*, 2001.

64. Radu Rugina. Quantitative shape analysis. In *Static Analysis Symposium (SAS'04)*, 2004.
65. Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24(3):217–298, 2002.
66. E. Schonberg, J. T. Schwartz, and M. Sharir. An automatic technique for selection of data representations in Setl programs. *Transactions on Programming Languages and Systems*, 3(2):126–143, 1991.
67. 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.
68. Larry Stockmeyer and Albert R. Meyer. Cosmological lower bound on the circuit complexity of a small problem in logic. *J. ACM*, 49(6):753–784, 2002.
69. A. Stump, C. Barrett, and D. Dill. CVC: a Cooperating Validity Checker. In *14th International Conference on Computer-Aided Verification*, 2002.
70. Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages Vol.3: Beyond Words*. Springer-Verlag, 1997.
71. Cesare Tinelli. Cooperation of background reasoners in theory reasoning by residue sharing. *Journal of Automated Reasoning*, 30(1):1–31, January 2003.
72. Cesare Tinelli and Calogero Zarba. Combining non-stably infinite theories. *Journal of Automated Reasoning*, 2004. (Accepted for publication).
73. Ashish Tiwari. *Decision procedures in automated deduction*. PhD thesis, Department of Computer Science, State University of New York at Stony Brook, 2000.
74. John Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *Dublin Philosophical Magazine and Journal of Science*, 9(59):1–18, 1880.
75. Andrei Voronkov. The anatomy of Vampire (implementing bottom-up procedures with code trees). *Journal of Automated Reasoning*, 15(2):237–265, 1995.
76. Greta Yorsh, Thomas Reps, and Mooly Sagiv. Symbolically computing most-precise abstract operations for shape analysis. In *10th TACAS*, 2004.
77. Calogero G. Zarba. *The Combination Problem in Automated Reasoning*. PhD thesis, Stanford University, 2004.
78. Calogero G. Zarba. Combining sets with elements. In Nachum Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 762–782. Springer, 2004.
79. Calogero G. Zarba. A quantifier elimination algorithm for a fragment of set theory involving the cardinality operator. In *18th International Workshop on Unification*, 2004.
80. Karen Zee, Patrick Lam, Viktor Kuncak, and Martin Rinard. Combining theorem proving with static analysis for data structure consistency. In *International Workshop on Software Verification and Validation (SVV 2004)*, Seattle, November 2004.

APPENDIX

9 Proofs of Lemmas

Lemma 1 *Let $b_1, \ldots, b_n$ be finite disjoint sets, and $l_1, \ldots, l_n, k_1, \ldots, k_n$ be natural numbers. Then the following two statements are equivalent: (1) There exists a finite set y such that $\bigwedge_{i=1}^{n} |b_i \cap y| = k_i \wedge |b_i \cap y^c| = l_i$ and (2) $\bigwedge_{i=1}^{n} |b_i| = k_i + l_i$. Moreover, the statement continues to hold if for any subset of indices i the conjunct $|b_i \cap y| = k_i$ is replaced by $|b_i \cap y| \geq k_i$ or the conjunct $|b_i \cap y^c| = l_i$ is replaced by $|b_i \cap y^c| \geq l_i$, provided that $|b_i| = k_i + l_i$ is replaced by $|b_i| \geq k_i + l_i$, as indicated in Figure 10.*

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 $|b_i| = k_i + l_i$ when the conjuncts are

$|b_i \cap y| = k_i \wedge |b_i \cap y^c| = l_i$, and $|b_i| \geq k_i + l_i$ if any of the original conjuncts have inequality.

($\Leftarrow$) Suppose that (2) holds. First consider the case of equalities. Suppose that $|b_i| = k_i + l_i$ for each of the pairwise disjoint sets $b_1, \ldots, b_n$. 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 $y_1, \ldots, y_n$, 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. The case of inequalities is analogous: for example, in the case $|b_i \cap y| \geq k_i \wedge |b_i \cap y^c| = l_i$, choose $y_i \subseteq b_i$ such that $|y_i| = |b_i| - l_i$.

Lemma 3. *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 $2^{O(\mathsf{size}(F_0))}$ space.*

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 11 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 certainly $O(2^{cn})$ for any $c > 1$. Moreover, 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.

Lemma 5. *For each r where $1 \leq r \leq S$ the truth value of $G_r(w_1, \ldots, w_q)$ is equal to the the truth value of $G_r(\bar{w}_1, \ldots, \bar{w}_q)$ 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 $G_{r+1}(\bar{w}'_1, \ldots, \bar{w}'_{q'})$.

Suppose first that $G_{r+1}(\bar{w}'_1, \ldots, \bar{w}'_{q'})$ holds. Then for each w'_i there are w_{2i-1} and w_{2i} such that $\bar{w}'_i = u_{2i-1} + u_{2i}$ and $G_r(u_1, \ldots, u_q)$. We define witnesses $w_1, \ldots, w_q$ as follows. If $w'_i \leq 2^r$, then let $w_{2i-1} = u_{2i-1}$ and $w_{2i} = u_{2i}$. If $w'_i > 2^r$ then either

$u_{2i-1} > 2^{r-1}$ or $u_{2i} > 2^{r-1}$ (or both). If $u_{2i-1} > 2^{r-1}$, then let $w_{2i-1} = w'_i - u_{2i}$ and $w_{2i} = u_{2i}$. Note that $G_r(\ldots, w_{2i-1}, \ldots) \iff G_r(\ldots, u_{2i-1}, \ldots) \iff G_r(\ldots, 2^{r-1}, \ldots)$ by induction hypothesis because both $u_{2i-1} > 2^{r-1}$ and $w_{2i-1} > 2^{r-1}$. 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 $w_{2i} \leq w_{2i}$ 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 $w_{2i} > w_{2i}$ then let $u_{2i-1} = 2^{r-1}$ and $u_{2i} = 2^{r-1}$. 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 $G_r(u_1, \ldots, u_q) = G_r(w_1, \ldots, w_q)$. 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}.

10 BAPA with Potentially Infinite Sets

We next sketch the extension of our algorithm α (Section 4) to the case when the universe of the structure may be infinite, and the underlying language has the ability to distinguish between finite and infinite sets. Infinite sets are useful in program analysis for modelling pools of objects such as those arising in dynamic object allocation. This section presents an approach that avoids directly reasoning about cardinalities of infinite sets and thus remains within the language of PA. (As was observed in [19], an alternative is to use a generalization of PA that admits infinite cardinals.)

We generalize the language of BAPA and the interpretation of BAPA operations as follows.

1. Introduce unary predicate $\mathsf{fin}(b)$ which is true iff b is a finite set. The predicate $\mathsf{fin}(b)$ allows us to generalize our algorithm to the case of infinite universe, and additionally gives the expressive power to distinguish between finite and infinite sets. For example, using $\mathsf{fin}(b)$ we can express bounded quantification over finite or over infinite sets.
2. Define $|b|$ to be the integer zero if b is infinite, and the cardinality of b if b is finite.
3. Introduce propositional variables denoted by letters such as p, q, and quantification over propositional variables. Extend also the underlying PA formulas with propositional variables, which is acceptable because a variable p can be treated as a shorthand for an integer from $\{0, 1\}$ if each use of p as an atomic formula is interpreted as the atomic formula $(p = 1)$. Our extended algorithm uses the equivalences $\mathsf{fin}(b) \Leftrightarrow p$ to represent the finiteness of sets just as it uses the equations $|b| = l$ to represent the cardinalities of finite sets.
4. Introduce a propositional constant FINU such that $\mathsf{fin}(\mathcal{U}) \Leftrightarrow \mathsf{FINU}$. This propositional constant enables equivalence preserving quantifier elimination over the set of models that includes both models with finite universe $\mathcal{U}$ and the models with infinite universe $\mathcal{U}$.

Denote the resulting extended language BAPA^{∞}.

The following lemma generalizes Lemma 1 for the case of equalities.

Lemma 7. *Let $b_1, \ldots, b_n$ be disjoint sets, $l_1, \ldots, l_n, k_1, \ldots, k_n$ be natural numbers, and $p_1, \ldots, p_n, q_1, \ldots, q_n$ be propositional values. Then the following two statements are equivalent:*

1. *There exists a set y such that*

$$\bigwedge_{i=1}^{n} \begin{array}{l} |b_i \cap y| = k_i \wedge (\mathsf{fin}(b_i \cap y) \Leftrightarrow p_i) \wedge \\ |b_i \cap y^c| = l_i \wedge (\mathsf{fin}(b_i \cap y^c) \Leftrightarrow q_i) \end{array} \tag{7}$$

2.

$$\bigwedge_{i=1}^{n} \begin{array}{l} (p_i \wedge q_i \Rightarrow |b_i| = k_i + l_i) \wedge \\ (\mathsf{fin}(b_i) \Leftrightarrow (p_i \wedge q_i)) \end{array} \tag{8}$$

Proof. ($\Rightarrow$) Suppose that there exists a set y satisfying (7). From $b_i = (b_i \cap y) \cup (b_i \cap y^c)$, we have $\mathsf{fin}(b_i) \Leftrightarrow (p_i \wedge q_i)$. Furthermore, if p_i and q_i hold, then both $b_i \cap y$ and $b_i \cap y^c$ are finite so the relation $|b_i| = |b_i \cap y| + |b_i \cap y^c|$ holds.

($\Leftarrow$) Suppose that (8) holds. For each i we choose a subset $y_i \subseteq b_i$, depending on the truth values of p_i and q_i, as follows.

1. If both p_i and q_i are true, then $\mathsf{fin}(b_i)$ holds, so b_i is finite. Choose y_i as any subset of b_i with k_i elements, which is possible since b_i has $k_i + l_i$ elements.
2. If p_i does not hold, but q_i holds, then $\mathsf{fin}(b_i)$ does not hold, so b_i is infinite. Choose y_i' as any finite set with l_i elements and let $y_i = b_i \setminus y_i'$ be the corresponding cofinite set.
3. Analogously, if p_i holds, but q_i does not hold, then b_i is infinite; choose y_i as any finite subset of b_i with k_i elements.
4. If p_i and q_i are both false, then b_i is also infinite; every infinite set can be written as a disjoint union of two infinite sets, so let y_i be one such set.

Let $y = \bigcup_{i=1}^{n} y_i$. As in the proof of Lemma 1, we have $b_i \cap y = y_i$ and $b_i \cap y^c = y_i^c$. By construction of $y_1, \ldots, y_n$ we conclude that (7) holds.

The algorithm α for BAPA^∞ is analogous to the algorithm for BAPA. In each step, the new algorithm maintains a formula of the form

$$\begin{array}{l} Q_p v_p \ldots Q_r v_r. \\ \quad \exists^+ l_1 \ldots l_q.\ \exists p_1 \ldots p_q. \\ \quad\quad (\bigwedge_{i=1}^{q} |s_i| = l_i \wedge (\mathsf{fin}(s_i) \Leftrightarrow p_i)) \ \wedge\ G_r \end{array}$$

As in Section 4, the algorithm eliminates an integer quantifier $\exists k$ by letting $G_{r+1} = \exists k. G_r$ and eliminates an integer quantifier $\forall k$ by letting $G_{r+1} = \forall k. G_r$. Furthermore, just as the algorithm in Section 4 uses Lemma 1 to reduce a set quantifier to integer quantifiers, the new algorithm uses Lemma 7 for this purpose. The algorithm replaces

$$\begin{array}{l} \exists y.\ \exists^+ l_1 \ldots l_q.\ \exists p_1 \ldots p_q. \\ \quad\quad (\bigwedge_{i=1}^{q} |s_i| = l_i \wedge (\mathsf{fin}(s_i) \Leftrightarrow p_i)) \ \wedge\ G_r \end{array}$$

with

$$\begin{array}{l} \exists^+ l_1' \ldots l_{q'}'.\ \exists p_1' \ldots p_{q'}'. \\ \quad\quad (\bigwedge_{i=1}^{q'} |s_i'| = l_i' \wedge (\mathsf{fin}(s_i') \Leftrightarrow p_i')) \ \wedge\ G_{r+1} \end{array}$$

for $q' = q/2$, and

$$
\begin{aligned}
G_{r+1} \;\equiv\;& \exists^+ l_1 \ldots l_q.\, \exists p_1, \ldots, p_q. \\
& \Big(\textstyle\bigwedge_{i=1}^{q'} \, (p_{2i-1} \wedge p_{2i} \Rightarrow l'_i = l_{2i-1} + l_{2i}) \wedge \\
& \qquad (p'_i \Leftrightarrow (p_{2i-1} \wedge p_{2i})) \Big) \\
& \wedge\, G_r
\end{aligned}
$$

For the quantifier $\forall y$ the algorithm analogously generates

$$
\begin{aligned}
G_{r+1} \;\equiv\;& \forall^+ l_1 \ldots l_q.\, \forall p_1, \ldots, p_q. \\
& \Big(\textstyle\bigwedge_{i=1}^{q'} \, (p_{2i-1} \wedge p_{2i} \Rightarrow l'_i = l_{2i-1} + l_{2i}) \wedge \\
& \qquad (p'_i \Leftrightarrow (p_{2i-1} \wedge p_{2i})) \Big) \\
& \Rightarrow\; G_r
\end{aligned}
$$

After eliminating all quantifiers, the algorithm obtains a formula of the form $\exists^+ l.\exists p.\; |\mathcal{U}| = l \wedge (\mathsf{fin}(\mathcal{U}) \Leftrightarrow p) \wedge G_{p+1}(l, p)$. We define the result of the algorithm to be the PA sentence $G_{p+1}(\mathsf{MAXC}, \mathsf{FINU})$.

This completes our description of the generalized algorithm α for BAPA^∞. The complexity analysis from Section 5 also applies to the generalized version. We also note that our algorithm yields an equivalent formula over any family of models. A sentence is valid in a set of models iff it is valid on each model. Therefore, the validity of a BAPA^∞ sentence F_0 is given by applying to the formula $\alpha(F_0)(\mathsf{MAXC}, \mathsf{FINU})$ a form of universal quantifier over all pairs $(\mathsf{MAXC}, \mathsf{FINU})$ that determine the characteristics of the models in question. For example, for the validity over the models with infinite universe we use $\alpha(F_0)(0, \mathsf{false})$, for validity over all finite models we use $\forall k.\alpha(F_0)(k, \mathsf{true})$, and for the validity over all models we use the PA formula

$$\alpha(F_0)(0, \mathsf{false}) \wedge \forall k.\alpha(F_0)(k, \mathsf{true}).$$

We therefore have the following result.

Theorem 8. *The algorithm above effectively reduces the validity of* BAPA^∞ *sentences to the validity of Presburger arithmetic formulas with the same number of quantifier alternations, and the increase in formula size exponential in the number of set variables; the reduction works for each of the following: 1) the set of all models, 2) the set of models with infinite universe only, and 3) the set of all models with finite universe.*

11 Relationship with MSOL over Strings

The monadic second-order logic (MSOL) over strings is a decidable logic that can encode Presburger arithmetic by encoding addition using one successor symbol and quantification over sets. This logic therefore simultaneously supports sets and integers, so it is natural to examine its relationship with BAPA. It turns out that 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. In MSOL over strings, the sets contain binary digits of an integer whereas in BAPA the sets contain *uninterpreted elements*.

Given these differences, a natural question is to consider the decidability of an extension of MSOL that allows stating relations $|A| = k$ where A is a set of digits and k is an integer variable. Note that by saying $\exists k.|A| = k \wedge |B| = k$ we can express $|A| = |B|$, so we obtain MSOL with equicardinality constraints. However, extensions of MSOL over strings with equicardinality constraints are known to be undecidable; we review some reductions in Section 11.2. Undecidability results such as these are what perhaps led to the conjectures that BAPA itself is undecidable [79, Page 12]. In this paper we pointed out that BAPA is, in fact, decidable and proved that it has an elementary decision procedure. Moreover, we present a combination of BA with MSOL over n-successors that is still decidable.

11.1 Decidability of MSOL with Cardinalities on Uninterpreted Sets

We next note that our algorithm also applies to monadic second-order logic of one successors, which is more expressive than PA itself [70, Page 400], [10].

Consider the multisorted language BAMSOL defined as follows. First, BAMSOL contains all relations of monadic second-order logic of one successors, whose variables range over strings over an n-ary alphabet and sets of such strings. Second, BAMSOL contains sets of uninterpreted elements and boolean algebra operations on them. Third, BAMSOL allows stating relationships of the form $|x| = k$ where x is a set of uninterpreted elements and k is a string representing a natural number. Because all PA operations are definable in MSOL of 1-successor, the algorithm α applies in this case as well. Indeed, the algorithm α only needs a "lower bound" on the expressive power of the theory of integers that BA is combined with: the ability to state constraints of the form $l'_i = l_{2i-1} + l_{2i}$, and quantification over integers. Therefore, applying α to a BAMSOL formula results in an MSOL formula. This shows that BAMSOL is decidable and can be decided using a combination of algorithm α and a tool such as [31]. By Lemma 3, the decision procedure for BAMSOL based on translation to MSOL has upper bound of $\mathsf{exp}_n(O(n))$ using a decision procedure such as [31]. The corresponding non-elementary lower bound follows from the lower bound on MSOL itself [68].

11.2 Undecidability of MSOL of Integer Sets with Cardinalities

We first note that there is a reduction from the Post Correspondence Problem that shows the undecidability of MSOL with equicardinality constraints. Namely, we can represent binary strings by finite sets of natural numbers. In this encoding, given a position, MSOL itself can easily express the local property that, at a given position, a string contains a given finite substring. The equicardinality gives the additional ability of finding an n-th element of an increasing sequence of elements. To encode a PCP instance, it suffices to write a formula checking the existence of a string (represented as set A) and the existence of two increasing sequences of equal length (represented by sets U and D), such that for each i, there exists a pair (a_j, b_j) of PCP instance such that the position starting at U_i contains the constant string a_j, and $U_{i+1} = U_i + |a_j|$, and similarly the position starting at D_i contains b_j and $D_{i+1} = D_i + |b_j|$.

12 O'Caml source code of algorithm α

```
(* ------------------------------------------- *)
(*             datatype of formulas            *)

type ident = string

type binder =   Forallset | Existsset
              | Forallint | Existsint | Forallnat | Existsnat
type form =
  | Not of form
  | And of form list | Or of form list | Impl of form * form
  | Bind of binder * ident * form
  | Inteq of intTerm * intTerm | Less of intTerm * intTerm
  | Seteq of setTerm * setTerm | Subseteq of setTerm * setTerm
and intTerm =
  | Intvar of ident | Const of int
  | Plus of intTerm list | Minus of intTerm * intTerm | Times of int * intTerm
  | Card of setTerm
and setTerm =
  | Setvar of ident | Emptyset | Fullset | Complement of setTerm
  | Union of setTerm list | Inter of setTerm list

let maxcard = "MAXC"

(* ------------------------------------------- *)
(*             algorithm \alpha                *)

  (* replace Seteq and Subseteq with Card(...)=0 *)
let simplify_set_relations (f:form) : form =
  let rec sform f = match f with
  | Not f -> Not (sform f)
  | And fs -> And (List.map sform fs)
  | Or fs -> Or (List.map sform fs)
  | Impl(f1,f2) -> Impl(sform f1,sform f2)
  | Bind(b,id,f1) -> Bind(b,id,sform f1)
  | Less(it1,it2) -> Less(it1, it2)
  | Inteq(it1,it2) -> Inteq(it1,it2)
  | Seteq(st1,st2) -> And[sform (Subseteq(st1,st2));
                          sform (Subseteq(st2,st1))]
  | Subseteq(st1,st2) -> Inteq(Card(Inter[st1;Complement st2]),Const 0)
  in sform f

  (* split f into quantifier sequence and body *)
let split_quants_body f =
  let rec vl f acc = match f with
  | Bind(b,id,f1) -> vl f1 ((b,id)::acc)
  | f -> (acc,f)
  in vl f []

  (* extract set variables from quantifier sequence *)
let extract_set_vars quants =
  List.map (fun (b,id) -> id)
    (List.filter (fun (b,id) -> (b=Forallset || b = Existsset))
       quants)

type partition = (ident * setTerm) list

  (* make canonical name for integer variable naming a cube *)
let make_name sts =
  let rec mk sts = match sts with
  | [] -> ""
  | (Setvar _)::sts1 -> "1" ^ mk sts1
  | (Complement (Setvar _))::sts1 -> "0" ^ mk sts1
  | _ -> failwith "make_name: unexpected partition form"
  in "l_" ^ mk sts

  (* make all cubes over vs *)
let generate_partition (vs : ident list) : partition =
```

```
  let add id ss = (Setvar id)::ss in
  let addc id ss = Complement (Setvar id)::ss in
  let add_set id inters =
    List.map (add id) inters @
    List.map (addc id) inters in
  let mk_nm is = (make_name is, Inter is) in
  List.map mk_nm
    (List.map List.rev
       (List.fold_right add_set vs [[]]))

  (* is the set term true in the set valuation
     -- reduces to propositional reasoning *)
let istrue (st:setTerm) (id,ivaluation) : bool =
  let valuation = match ivaluation with
  | Inter v -> v
  | _ -> failwith "wrong valuation" in
  let lookup v =
    if List.mem (Setvar v) valuation then true
    else if List.mem (Complement (Setvar v)) valuation then false
    else failwith "istrue: unbound var in valuation" in
  let rec check st = match st with
  | Setvar v -> lookup v
  | Emptyset -> false
  | Fullset -> true
  | Complement st1 -> not (check st1)
  | Union sts -> List.fold_right (fun st1 t -> check st1 || t) sts false
  | Inter sts -> List.fold_right (fun st1 t -> check st1 && t) sts true
  in check st

  (* compute cardinality of set expression
     as a sum of cardinalities of cubes *)
let get_sum (p:partition) (st:setTerm) : intTerm list =
  let get_list (id,inter) = match inter with
  | Inter ss -> ss
  | _ -> failwith "failed inv in get_sum"
  in
  List.map (fun (id,inter) -> Intvar id)
    (List.filter (istrue st) p)

  (* replace cardinalities of sets with sums of
     variables denoting cube cardinalities *)
let replace_cards (p:partition) (f:form) : form =
  let rec repl f = match f with
  | Not f -> Not (repl f)
  | And fs -> And (List.map repl fs)
  | Or fs -> Or (List.map repl fs)
  | Impl(f1,f2) -> Impl(repl f1,repl f2)
  | Bind(b,id,f1) -> Bind(b,id,repl f1)
  | Less(it1,it2) -> Less(irepl it1,irepl it2)
  | Inteq(it1,it2) -> Inteq(irepl it1,irepl it2)
  | Seteq(_,_)|Subseteq(_,_) -> failwith "failed inv in replace_cards"
  and irepl it = match it with
  | Intvar _ -> it
  | Const _ -> it
  | Plus its -> Plus (List.map irepl its)
  | Minus(it1,it2) -> Minus(irepl it1, irepl it2)
  | Times(k,it1) -> Times(k, irepl it1)
  | Card st -> Plus (get_sum p st)
  in repl f

let apply_quants quants f =
  List.fold_right (fun (b,id) f -> Bind(b,id,f)) quants f

let make_defining_eqns id part =
  let rec mk ps = match ps with
  | [] -> []
  | (id1,Inter (st1::sts1)) :: (id2,Inter (st2::sts2)) :: ps1
    when (st1=Setvar id && st2=Complement (Setvar id) && sts1=sts2) ->
      (Inter sts1,make_name sts1,id1,id2) :: mk ps1
```

```
  | _ -> failwith "make_triples: unexpected partition form" in
  let rename_last lss = match lss with
  | [(s,l,l1,l2)] -> [(s,maxcard,l1,l2)]
  | _ -> lss in
  rename_last (mk part)

  (* ------------------------- *)
  (* main loop of the algorithm *)

let rec eliminate_all quants part gr = match quants with
  | [] -> gr
  | (Existsint,id)::quants1 ->
      eliminate_all quants1 part (Bind(Existsint,id,gr))
  | (Forallint,id)::quants1 ->
      eliminate_all quants1 part (Bind(Forallint,id,gr))
  | (Existsnat,id)::quants1 ->
      eliminate_all quants1 part (Bind(Existsnat,id,gr))
  | (Forallnat,id)::quants1 ->
      eliminate_all quants1 part (Bind(Forallnat,id,gr))
  | (Existsset,id)::quants1 ->
      let eqns = make_defining_eqns id part in
      let newpart = List.map (fun (s,l',_,_) -> (l',s)) eqns in
      let mk_conj (_,l',l1,l2) = Inteq(Intvar l',Plus[Intvar l1;Intvar l2]) in
      let conjs = List.map mk_conj eqns in
      let lquants = List.map (fun (l,_) -> (Existsnat,l)) part in
      let gr1 = apply_quants lquants (And (conjs @ [gr])) in
      eliminate_all quants1 newpart gr1
  | (Forallset,id)::quants1 ->
      let eqns = make_defining_eqns id part in
      let newpart = List.map (fun (s,l',_,_) -> (l',s)) eqns in
      let mk_conj (_,l',l1,l2) = Inteq(Intvar l',Plus[Intvar l1;Intvar l2]) in
      let conjs = List.map mk_conj eqns in
      let lquants = List.map (fun (l,_) -> (Forallnat,l)) part in
      let gr1 = apply_quants lquants (Impl(And conjs, gr)) in
      eliminate_all quants1 newpart gr1

(* putting everything together *)

let alpha (f:form) : form =
  (* assumes f in prenex form *)
  let (quants,fm) = split_quants_body f in
  let fm1 = simplify_set_relations fm in
  let setvars = List.rev (extract_set_vars quants) in
  let part = generate_partition setvars in
  let g1 = replace_cards part fm1 in
  eliminate_all quants part g1
```