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 a priory unbounded finite sets, and supports arbitrary quantification over 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 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, and have applications in constraint databases.

We give a formal description of a decision procedure for BAPA, which implies the decidability of 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 an optimal decision procedure for BA through 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 [18] 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 [29] we identify the first-order theory of Boolean algebras (BA) as a

Compiled June 19, 2005, 12:14pm. Updated version of CADE-20 (typos fixed).

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. (We are interested in BA of all subsets of some set; this theory was shown decidable already in [31,46], see [22] for the discussion of other models of Boolean algebra axioms.)

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 [14] (which also proves the well-known Feferman-Vaught theorem [19, 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 [55], cast as a multi-sorted theory. Starting from [29] as our motivation, we have observed in [26] the decidability of the full BAPA (which was initially left open in [55]). An algorithm for a single-sorted version of BAPA was presented independently in [42] as a way of evaluating queries in constraint databases; [42] leaves open the complexity of the satisfiability 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 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.
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 initial experience in using our **implementation** of BAPA to discharge verification conditions generated in the Jahob verification system [23].
5. In addition, we note the following related results:
   (a) PA sentences generated by translating BA sentences without cardinalities can be decided in **optimal** alternating time (Section 5.2);
   (b) Our algorithm extends to **countable sets** with a predicate distinguishing finite and infinite sets (Section 7);
   (c) In contrast to the undecidability of MSOL with equicardinality operator, we identify a **decidable** combination of MSOL over trees with BA (Section 7).

A preliminary version of our results, including the algorithm and complexity analysis appear in [26], which also contains proofs and further details of our results.

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 [28]. 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 [31, 46]. 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 $A \sim B$ 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 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 [14, 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.

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 3, and contains

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

Fig. 1. Formulas of Boolean Algebra (BA)

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

Fig. 2. Formulas of Presburger Arithmetic (PA)

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

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

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 7.

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.

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 [13] or can arise as abstractions of programs in standard languages [29]. 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

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 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.

$\mathbf{var}\ \mathsf{R} : \mathbf{set};$
$\mathbf{var}\ \mathsf{S} : \mathbf{set};$

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

$\mathbf{var}\ \mathsf{P} : \mathbf{set};$
$\mathbf{var}\ \mathsf{k} : \mathbf{integer};$

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

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 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 state into a non-negative integer, 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 belongs to BAPA, and can

```
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

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 [38]. 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 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 the space and time bounds for PA sentences [41], 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 [21, 30].
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 7.
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

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

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 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). The cubes have the form $\bigwedge_{i=1}^{n} x_i^{\alpha_i}$ 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. Then the resulting formula is equivalent to

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

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

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

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 $Q_r v_r$.

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

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

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

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 (see [26] for proof).

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 $\bigwedge_{i=1}^{n} |b_i \cap y| = k_i \wedge |b_i \cap y^c| = l_i$*
2. *$\bigwedge_{i=1}^{n} |b_i| = k_i + l_i$.*

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

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

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

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

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

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 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. 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

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

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 [39], 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 wrote a concise implementation in O'Caml, see [26]. 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 10. 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 11 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 11.

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

Fig. 10. The translation of the BAPA sentence from Figure 6 into a PA sentence

general relationship:

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

in this example:

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

Fig. 11. The Correspondence between Integer Variables in Figure 10 and Set Variables in Figure 6

5 Complexity

In this section we analyze the algorithm α from Section 4 and obtain space bounds on BAPA from the corresponding space bounds for PA. We then show that the new decision procedure is optimal for BA if applied to BA formulas. Moreover, by construction, our procedure reduces to the procedure for PA formulas if there are no set quantifiers. In summary, our decision procedure is optimal 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.

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 exponential 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 $\mathsf{exp}_{k+1}(x) = 2^{\mathsf{exp}_k(x)}$.

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 $2^{O(\mathsf{size}(F_0))}$ time and space.*

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

Fact 1 *[15, 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 *[41] 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 PA 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 has the same number of exponentials as the bound for PA.

5.2 BA as a Special Case

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 best upper bounds for PA [15, Chapter 3] are worse than the corresponding bounds for BA [22]. However, we identify a special form of PA sentences $\mathsf{PA}_{\mathsf{BA}} = \{\alpha(F_0) \mid F_0 \text{ is in BA}\}$ and show that such sentences can be decided in alternating time optimal for BA [22].

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 5. *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 $\bar{w}_i = \min(w_i, 2^{r-1})$.*

Consider a formula F_0 of size n with S 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 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 $2^{S+1} - 1$ variables can be represented in $O(S)$ space. Because $\alpha(F_0)$ has S quantifier alternations, $\alpha(F_0)$ the values of all bound variables can be guessed in alternating time $O(S)$. 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 alternating time bounded by $n^a 2^{bS}$ for some constants a, b. Because $S \leq n$, we conclude that the algorithm α can be used to decide a pure BA formula by alternating Turing machine running in time 2^{cn} for some $c > 0$ and performing n alternations. The class of all such problems is called Berman complexity class $\mathsf{STA}(*, 2^{cn}, n)$. Theorem 5.6 in [22] shows that BA (even if interpreted only over all finite Boolean algebras) is in fact complete for the class $\mathsf{STA}(*, 2^{cn}, n)$. Therefore, our algorithm α allows optimal decision procedure for BA, if the PA decision procedure exploits the special structure of the generated formula $\alpha(F_0)$; this special structure is given by Lemma 5. Note that the class $\mathsf{STA}(*, 2^{cn}, n)$ is contained in the deterministic exponential space, which is equal to alternating exponential time, the only difference being that the number of alternations in $\mathsf{STA}(*, 2^{cn}, n)$ is restricted to be linear.

6 Experience Using Our Decision Procedure for BAPA

We have experimented with BAPA in the context of Jahob system [23] 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 [36], the Simplify theorem prover [12] as well as the Omega Calculator [40] and the LASH [30] 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, 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. The formulas arising from examples in Section 3 have also been discharged using our current implementation. By comparing different decision procedures, 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 initial experience indicates that the direct implementation of the basic algorithm 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, less than 10 seconds for 5 variables. More than 5 set variables cause the PA decision procedure to run out of memory. (We have used the Omega Calculator to decide PA formulas because we found that it outperforms LASH in 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 system 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 perform formula simplifications during this process [26, Section 5.2]; this alternative approach does not yield good worse-case complexity bounds but could be useful for subclasses of BAPA formulas.

7 Further Observations

We next sketch some further observations about BAPA, see [26] for details.

Countable sets. A generalization of BAPA where set variables range over subets of an arbitrary (not necessarily finite) set is decidable, which follows from the decidability of the first-order theory of the addition of cardinals [14]. We here consider the case of all subsets of a countable set, and argue that the complexity results we have developed so far still apply. We first generalize the language of BAPA and the interpretation of BAPA operations, as follows. Introduce function $\mathsf{inf}(b)$ which returns 0 if b is a finite set and 1 if b is a countable 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 finite cardinal can therefore be represented in PA using a pair (k, i) of an integer k and an infinity flag i. The relation representing the addition of cardinals $(k_1, i_1) + (k_2, i_2) = (k_3, i_3)$ is then definable by formula

$$(i_1 = 0 \land i_2 = 0 \land i_3 = 0 \land k_1 + k_2 = k_3) \;\lor\; ((i_1 \neq 0 \lor i_2 \neq 0) \land i_3 = 1 \land k_3 = 0)$$

Moreover, we have the following generalization of Lemma 1.

Lemma 6. *Let $b_1, \ldots, b_n$ be disjoint sets, $l_1, \ldots, l_n, k_1, \ldots, k_n$ be natural numbers, and $p_1, \ldots, p_n, q_1, \ldots, q_n \in \{0, 1\}$. Then the following two statements are equivalent:*

1. *There exists a set y such that*
$$\bigwedge_{i=1}^{n} |b_i \cap y| = k_i \land \mathsf{inf}(b_i \cap y) = p_i \;\land\; |b_i \cap y^c| = l_i \land \mathsf{inf}(b_i \cap y^c) = q_i$$
2. $$\bigwedge_{i=1}^{n} (p_i = 0 \land q_i = 0 \Rightarrow |b_i| = k_i + l_i) \;\land\; (\mathsf{inf}(b_i) = 0 \Leftrightarrow (p_i = 0 \land q_i = 0))$$

The algorithm for the case of countable set then generalizes using Lemma 6 in the natural way; the resulting PA formulas are at most polynomially larger than for the finite case, so we obtain the same complexity bounds.

Relationship to MSOL. 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. 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 undecidabe, 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, so it implies the decidability of the BAPA extension where the constraints on cardinalities of sets are expressed using relations on integers definable in MSOL over strings; these relations go beyond PA [48, Page 400], [7].

8 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 shown in [14] (see Section 2). A decision procedure for a special case of BAPA was presented in [55], which allows only quantification over *elements* but not over *sets* of elements. [42] shows 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 efficiency reasons to allow integer variables.

Presburger arithmetic. The original result on decidability of PA is [39]. The best known bound on formula size is [15]. An analysis based on the number of quantifier alternations is presented in [41]. Our implementation uses quantifer-elimination based Omega test [40]. Among the decision procedures for full PA, [9] is the only proof-generating version, and is based on [11]. Decidable fragments of arithmetic that go beyond PA include [6, 21].

Boolean Algebras. The first results on decidability of BA are from [31], [1, Chapter 4] and use quantifier elimination, from which one can derive small model property; [22] gives the complexity of the satisfiability problem. [33] studies unification in Boolean rings. The quantifier-free fragment of BA is shown NP-complete in [32]; see [27] for a generalization of this result using parameterized complexity of the Bernays-Schönfinkel-Ramsey class of first-order logic [5, Page 258]. [8] 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. Among the systems for interactively reasoning about richer theories of sets are Isabelle [36], HOL [17], PVS [37], TPS [2]; first-order frameworks such as Athena [3] can use axiomatizations of sets along with calls to resolution-based theorem provers such as Vampire [51] to reason about sets.

Combinations of Decidable Theories. The techniques for combining *quantifier-free* theories [35,43] and their generalizations such as [49,50,53,54] 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 [14] and term powers [24, 25]. 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 [16, 47], which would likely improve the efficiency on common verification conditions compared to the current direct use of Omega decision procedure. Description logics [4] support sets with cardinalities as well as relations, but do not support 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 [34, 45, 52]. Recently, these approaches were extended to handle the combinations of the constraints representing data structure contents and constraints representing numerical properties of data structures [10,44]. 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. In addition to applications in Section 3, possible applications of our decision procedure include query evaluation in constraint databases [42] and loop invariant inference [20].

9 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 discharge verification conditions. 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, Chin Wei-Ngan, Calogero Zarba, Peter Revesz, Andreas Podelski, Bruno Courcelle, Cesare Tinelli, Konstantin Korovin, Stanford REACT group, Berkeley CHESS group, and CADE-20 reviewers on useful comments.

References

1. W. Ackermann. *Solvable Cases of the Decision Problem*. North Holland, 1954.
2. P. B. Andrews, S. Issar, D. Nesmith, and F. Pfenning. The TPS theorem proving system. In *10th CADE*, volume 449 of *LNAI*, pages 641–642, 1990.
3. K. Arkoudas, K. Zee, V. Kuncak, and M. 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.
4. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. CUP, 2003.
5. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.
6. M. Bozga and R. Iosif. On decidability within the arithmetic of addition and divisibility. In *FOSSACS'05*, 2005.
7. 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.

8. D. Cantone, E. Omodeo, and A. Policriti. *Set Theory for Computing*. Springer, 2001.
9. A. Chaieb and T. Nipkow. Generic proof synthesis for Presburger arithmetic. Technical report, Technische Universität München, October 2003.
10. W.-N. Chin, S.-C. Khoo, and D. N. Xu. Extending sized types with with collection analysis. In *ACM PEPM'03*, 2003.
11. 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.
12. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Laboratories Palo Alto, 2003.
13. R. K. Dewar. Programming by refinement, as exemplified by the SETL representation sublanguage. *ACM TOPLAS*, July 1979.
14. S. Feferman and R. L. Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.
15. J. Ferrante and C. W. Rackoff. *The Computational Complexity of Logical Theories*, volume 718 of *Lecture Notes in Mathematics*. Springer-Verlag, 1979.
16. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *16th CAV*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.
17. 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.
18. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
19. W. Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1993.
20. D. Kapur. Automatically generating loop invariants using quantifier elimination. In *IMACS Intl. Conf. on Applications of Computer Algebra*, 2004.
21. N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA implementation secrets. In *Proc. 5th International Conference on Implementation and Application of Automata*. LNCS, 2000.
22. D. Kozen. Complexity of boolean algebras. *Theoretical Computer Science*, 10:221–247, 1980.
23. V. Kuncak. The Jahob project web page. http://www.mit.edu/∼vkuncak/projects/jahob/, 2004.
24. V. Kuncak and M. Rinard. On the theory of structural subtyping. Technical Report 879, Laboratory for Computer Science, Massachusetts Institute of Technology, 2003.
25. V. Kuncak and M. Rinard. Structural subtyping of non-recursive types is decidable. In *Eighteenth Annual IEEE Symposium on Logic in Computer Science*, 2003.
26. V. Kuncak and M. Rinard. The first-order theory of sets with cardinality constraints is decidable. Technical Report 958, MIT CSAIL, July 2004.
27. V. Kuncak and M. Rinard. Decision procedures for set-valued fields. In *1st International Workshop on Abstract Interpretation of Object-Oriented Languages (AIOOL 2005)*, 2005.
28. P. Lam, V. Kuncak, and M. Rinard. Generalized typestate checking using set interfaces and pluggable analyses. *SIGPLAN Notices*, 39:46–55, March 2004.
29. P. Lam, V. Kuncak, and M. Rinard. Generalized typestate checking for data structure consistency. In *6th International Conference on Verification, Model Checking and Abstract Interpretation*, 2005.
30. LASH. The LASH Toolset. `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.
31. L. Loewenheim. Über mögligkeiten im relativkalkül. *Math. Annalen*, 76:228–251, 1915.
32. K. Marriott and M. Odersky. Negative boolean constraints. Technical Report 94/203, Monash University, August 1994.

33. U. Martin and T. Nipkow. Boolean unification: The story so far. *Journal of Symbolic Computation*, 7(3):275–293, 1989.
34. A. Møller and M. I. Schwartzbach. The Pointer Assertion Logic Engine. In *Proc. ACM PLDI*, 2001.
35. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM TOPLAS*, 1(2):245–257, 1979.
36. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.
37. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th CADE*, volume 607 of *LNAI*, pages 748–752, jun 1992.
38. A. Podelski and A. Rybalchenko. Transition predicate abstraction and fair termination. In *ACM POPL*, 2005.
39. 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.
40. W. 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.
41. C. R. Reddy and D. W. Loveland. Presburger arithmetic with bounded quantifier alternation. In *ACM STOC*, pages 320–325. ACM Press, 1978.
42. P. 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*, September 2004.
43. H. Ruess and N. Shankar. Deconstructing Shostak. In *Proc. 16th IEEE LICS*, 2001.
44. R. Rugina. Quantitative shape analysis. In *Static Analysis Symposium (SAS'04)*, 2004.
45. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24(3):217–298, 2002.
46. T. 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.
47. A. Stump, C. Barrett, and D. Dill. CVC: a Cooperating Validity Checker. In *14th International Conference on Computer-Aided Verification*, 2002.
48. W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages Vol.3: Beyond Words*. Springer-Verlag, 1997.
49. C. Tinelli and C. Zarba. Combining non-stably infinite theories. *Journal of Automated Reasoning*, 2004. (Accepted for publication).
50. A. Tiwari. *Decision procedures in automated deduction*. PhD thesis, Department of Computer Science, State University of New York at Stony Brook, 2000.
51. A. Voronkov. The anatomy of Vampire (implementing bottom-up procedures with code trees). *Journal of Automated Reasoning*, 15(2):237–265, 1995.
52. G. Yorsh, T. Reps, and M. Sagiv. Symbolically computing most-precise abstract operations for shape analysis. In *10th TACAS*, 2004.
53. C. G. Zarba. *The Combination Problem in Automated Reasoning*. PhD thesis, Stanford University, 2004.
54. C. G. Zarba. Combining sets with elements. In N. Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 762–782. Springer, 2004.
55. C. G. Zarba. A quantifier elimination algorithm for a fragment of set theory involving the cardinality operator. In *18th International Workshop on Unification*, 2004.