MUNCH - Automated Reasoner for Sets and Multisets

Ruzica Piskac and Viktor Kuncak

Swiss Federal Institute of Technology Lausanne (EPFL)
firstname.lastname@epfl.ch

Abstract. This system description provides an overview of the MUNCH reasoner for sets and multisets. MUNCH takes as the input a formula in a logic that supports expressions about sets, multisets, and integers. Constraints over collections and integers are connected using the cardinality operator. Our logic is a fragment of logics of popular interactive theorem provers, and MUNCH is the first fully automated reasoner for this logic. MUNCH reduces input formulas to equisatisfiable linear integer arithmetic formulas. MUNCH reasoner is publicly available. It is implemented in the Scala programming language and currently uses the SMT solver Z3 to solve the generated integer linear arithmetic constraints.

1 Introduction

Applications in software verification and interactive theorem proving often involve reasoning about sets of objects. Cardinality constraints on such collections also arise in these scenarios. Multisets arise for analogous reasons as sets: abstracting the content of linked data structure with duplicate elements leads to multisets. Multisets (and sets) are widely present in the theorem proving community. Interactive theorem provers such as Isabelle [3], Why [1] or KIV [4] specify theories of multisets with cardinality constraints. They prove a number of theorems about multisets to enable their use in interactive verification. However, all those tools require a certain level of interaction. Our tool is the first automated theorem prover for multisets with cardinality constraints, which can check satisfiability of formulas belonging to a very expressive logic (defined in Figure 1) entirely automatically.

This system description presents the implementation of the decision procedure for satisfiability of multisets with cardinality constraints [8]. We evaluated our implementation by checking the unsatisfiability of negations of verification conditions for the correctness of mutable data structure implementations. If an input formula is satisfiable, our tool generates a model, which can be used to construct a counterexample trace of the checked program.

2 Description of the MUNCH Implementation

2.1 Input Language

A multiset (bag) is a function m from a fixed finite set E to $\mathbb{N}$, where $m(e)$ denotes the number of times an element e occurs in the multiset (multiplicity of e). Our logic includes multiset operations such as multiplicity-preserving union and the intersection. In addition, our logic supports an infinite family of relations on multisets defined point-wise, one relation for each Presburger arithmetic formula. For example, $(m_1 \cap m_2)(e) = \min(m_1(e), m_2(e))$ and $m_1 \subseteq m_2$ means $\forall e. m_1(e) \leq m_2(e)$. Our logic supports using such point-wise operations for arbitrary quantifier-free Presburger arithmetic formulas. The logic also supports the cardinality operator that returns the number of elements in a multiset. Figure 1 summarizes the language of multisets with cardinality constraints (MAPA). There are two levels at which integer linear arithmetic constraints occur: to define point-wise operations on multisets (inner formulas) and to define constraints on cardinalities of multisets (outer formulas). Integer variables from outer formulas cannot occur within inner formulas.

Top-level formulas:

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

Outer linear arithmetic formulas:

$$\mathsf{F}_{\mathsf{out}} ::= \mathsf{A}_{\mathsf{out}} \mid \mathsf{F}_{\mathsf{out}} \wedge \mathsf{F}_{\mathsf{out}} \mid \neg \mathsf{F}_{\mathsf{out}}$$

$$\mathsf{A}_{\mathsf{out}} ::= \mathsf{t}_{\mathsf{out}} \leq \mathsf{t}_{\mathsf{out}} \mid \mathsf{t}_{\mathsf{out}}{=}\mathsf{t}_{\mathsf{out}} \mid (\mathsf{t}_{\mathsf{out}}, \ldots, \mathsf{t}_{\mathsf{out}}){=}\textstyle\sum_{\mathsf{F}^{\mathsf{in}}}(\mathsf{t}^{\mathsf{in}}, \ldots, \mathsf{t}^{\mathsf{in}})$$

$$\mathsf{t}_{\mathsf{out}} ::= k \mid C \mid \mathsf{t}_{\mathsf{out}} + \mathsf{t}_{\mathsf{out}} \mid C \cdot \mathsf{t}_{\mathsf{out}} \mid \mathsf{ite}(\mathsf{F}_{\mathsf{out}}, \mathsf{t}_{\mathsf{out}}, \mathsf{t}_{\mathsf{out}}) \mid \; | M |$$

Inner linear arithmetic formulas:

$$\mathsf{F}^{\mathsf{in}} ::= \mathsf{t}^{\mathsf{in}} \leq \mathsf{t}^{\mathsf{in}} \mid \mathsf{t}^{\mathsf{in}}{=}\mathsf{t}^{\mathsf{in}} \mid \mathsf{F}^{\mathsf{in}} \wedge \mathsf{F}^{\mathsf{in}} \mid \neg \mathsf{F}^{\mathsf{in}}$$

$$\mathsf{t}^{\mathsf{in}} ::= m(e) \mid C \mid \mathsf{t}^{\mathsf{in}} + \mathsf{t}^{\mathsf{in}} \mid C \cdot \mathsf{t}^{\mathsf{in}} \mid \mathsf{ite}(\mathsf{F}^{\mathsf{in}}, \mathsf{t}^{\mathsf{in}}, \mathsf{t}^{\mathsf{in}})$$

Multiset expressions:

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

C - integer constant Variables: e - fixed index, k - integer, m - multiset

Fig. 1. Quantifier-Free Multiset Constraints with Cardinality Operator (MAPA)

This logic subsumes the BAPA logic [5]. If a formula reasons only about sets, this can be added by explicitly stating for each set variable S that it is a set: $\forall e.(S(e) = 0 \vee S(e) = 1)$.

2.2 NP vs NEXPTIME Algorithm in Implementations

Checking satisfiability of MAPA formulas is an NP-complete problem [9]. Our first implementation was based on the algorithm used to establish this optimal complexity, but we found that the running times were impractical due to large constants. MUNCH therefore currently uses the conceptually simpler algorithm

in [8]. Despite its NEXPTIME worst-case complexity, we have found that the algorithm from [8], when combined with additional simplifications, results in a tool that exhibits acceptable performance. Our implementation often avoids the worst-case complexity of the most critical task, the computation of semilinear sets, by leveraging the special structure of formulas that we need to process (see Section 2.4).

2.3 System Overview

Figure 2 provides a high-level overview of the reasoner.

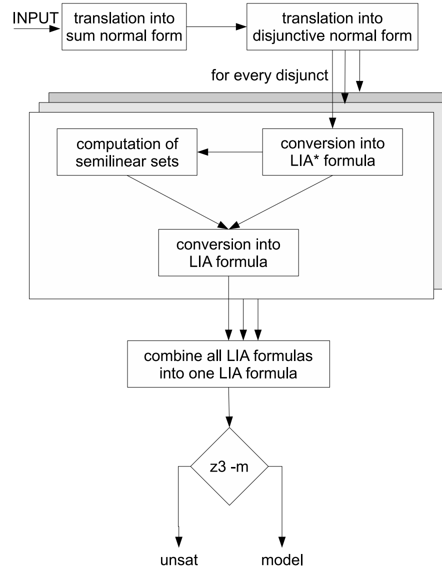

Fig. 2. Phases in checking formula satisfiability. MUNCH translates the input formula through several intermediate forms, preserving satisfiability in each step.

Given an input formula (Figure 1), MUNCH converts it into the *sum normal form*

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

where

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

The algorithm that reduces a formula to its sum normal form runs in polynomial time and is applicable to every input formula.

The derived formula is further translated into the logic that we call LIA^* [9]. LIA^* is linear integer arithmetic extended with the * operator. The * operator is defined on sets of vectors by $C^* = \{\boldsymbol{v}_1 + \ldots + \boldsymbol{v}_n \mid \boldsymbol{v}_1, \ldots \boldsymbol{v}_n \in C \wedge n \geq 0\}$. The new atom that we add to the linear integer arithmetic syntax is $\boldsymbol{u} \in \{\boldsymbol{x} \mid F(\boldsymbol{x})\}^*$, where F is a Presburger arithmetic formula.

A formula in the sum normal form

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

is equisatisfiable with the formula

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

where the terms t'_i and the formula F' are formed from the terms t_i and the formula F in the following way: for each multiset expression $m_j(e)$ we introduce a fresh new integer variable x_j and then we substitute each occurrence of $m_j(e)$ in the terms t_i and the formula F with the corresponding variable x_j. The equisatisfiability theorem between the two formulas follows from the definitions. Given a finite set of vectors $(t'_1, \ldots, t'_n)$ such that their sum is $(u_1, \ldots, u_n)$, we define the carrier set E to have as many elements as there are summands and define $m_j(e)$ to have the value of x_j in the corresponding summand. We use this theorem in the model reconstruction.

Model Reconstruction. Our tool outputs a model if an input formula is satisfiable. After all transformations, we obtain a linear arithmetic formula equisatisfiable to the input formula. If there is a satisfying assignment for the final formula, we use the constructive proofs of the equisatisfiability theorems to construct a model for the original formula.

2.4 Efficient Computation of Semilinear Sets and Elimination of the * Operator

The elimination of the * operator is done using semilinear sets. Let $S \subseteq \mathbb{Z}^m$ be a set of integer vectors and let $\boldsymbol{a} \in \mathbb{Z}^m$ be a integer vector. A linear set $LS(\boldsymbol{a}; S)$ is defined as $LS(\boldsymbol{a}; S) = \{\boldsymbol{a} + \boldsymbol{x}_1 + \ldots + \boldsymbol{x}_n \mid x_i \in S \wedge n \geq 0\}$. Note that vectors $\boldsymbol{x}_j$ and $\boldsymbol{x}_j$ can be equal and this way we can define a multiplication of a vector with a positive integer constant. A semilinear set Z is defined as a finite union of linear sets: $Z = \cup_{i=1}^k LS(a_i; S_i)$.

All vectors belonging to a semilinear set can be described as a solution set of a Presburger arithmetic formula. A classic result [2] shows that the converse also holds: the set of satisfying assignments of a Presburger arithmetic formula is a semilinear set.

Consider the set $\{(t'_1, \ldots, t'_n) \mid F'\}^*$. The set of all vectors which are the solution of formula F' is a semilinear set. It was shown in in [8,6] that applying

the * operator on a semilinear set results in a set which can be described by a Presburger arithmetic formula. Consequently, applying the star operator on a semilinear set results in a new semilinear set. Because $\{(t'_1, \ldots, t'_n) \mid F'\}^*$ is a semilinear set, checking whether $(u_1, \ldots, u_n) \in \{(t'_1, \ldots, t'_n) \mid F'\}^*$ is effectively expressible as a Presburger arithmetic formula. This concludes that the elimination of the * operator results in a equisatisfiable Presburger arithmetic formula.

Efficient Computation of Semilinear Sets The problem with this approach is that computing semilinear sets is expensive. The best know algorithms still run in the exponential time and are fairly complex [10].

For complexity reasons, we are avoiding to compute semilinear sets. Still, the exponential running time is unavoidable in this approach. Therefore, instead of developing an algorithm which computes semilinear sets for an arbitrary Presburger arithmetic formula, we split a formula into simpler parts for which we can easier compute semilinear sets. Namely, we convert formula F into a disjunctive normal form: $F'(\boldsymbol{t}) \equiv A_1(\boldsymbol{t}) \vee \ldots \vee A_m(\boldsymbol{t})$. This way, checking whether $\boldsymbol{u} \in \{\boldsymbol{t} \mid F'(\boldsymbol{t})\}^*$ reduces to $\boldsymbol{u} = \boldsymbol{k}_1 + \ldots + \boldsymbol{k}_m \wedge \bigwedge_{j=1}^{m} \boldsymbol{k}_j \in \{\boldsymbol{t} \mid A_j(\boldsymbol{t})\}^*$. The next task is to eliminate the $*$ operator for the formula $\boldsymbol{k}_j \in \{\boldsymbol{t} \mid A_j(\boldsymbol{t})\}^*$, where A_j is a conjunction of linear arithmetic atoms. A_j can also be rewritten as a conjunction of equalities by introducing fresh variables. In most of the cases, computing a semilinear set is actually computing a linear set which can be done effectively, for example, using the Omega-test [11]. Since A_j is a conjunction of equations, we use simple rewriting rules. This approach also supports certain inequalities. As an illustration, consider formula $m_0 = y + x$, where all variables have to be non-negative. All solutions are described with a linear set: $(m_0, y, x) = (0, 0, 0) + y(1, 1, 0) + x(1, 0, 1)$, i.e. $LS((0, 0, 0), \{(1, 1, 0), (1, 0, 1)\})$. This approach of using equalities and rewriting is highly efficient and works in most of cases. We also support a simple version of the Omega test.

However, our implementation is not complete for the full logic described in Figure 1. There are cases where one cannot avoid the computation of a semilinear set. One of the examples where the MUNCH tool cannot find a solution is when there exists an inner formula of the form $\forall e.F_{in}(e)$ and $F_{in}(e)$ is a formula where none of the variables have coefficient 1. An example of such a formula is $\forall e.5m_1(e) + 7m_2(e) \leq 6m_3(e)$. If at least one variable has coefficient 1 after the simplifications, our tool works. In our experimental results, while processing formulas derived in verification, we did not encounter such a problem. Notice also that our tool is always complete for sets, so it can also be used as a complete reasoner for sets with cardinality constraints (with a doubly exponential worst-case bound on running time).

To summarize, out of each conjunct we derive an equisatisfiable Presburger arithmetic formula and this way the initial multiset constraints problem reduces to satisfiability of quantifier-free Presburger arithmetic formulas. To check satisfiability of such a formula, we invoke the SMT solver Z3 [7] with the option "-m". This option ensures that Z3 returns a model in case that the input formula is satisfiable. Since all our transformations are satisfiability preserving, we either

return `unsat` or reconstruct a model for the initial multiset formula from the model returned by Z3.

3 Examples and Benchmarks

First we illustrate how the MUNCH reasoner works on a simple example, and then we show some benchmarks that we did.

Consider a simple multiset formula $|x \uplus y| = |x| + |y|$. Its validity is proved by showing that $|x \uplus y| \neq |x| + |y|$ is unsatisfiable. We chose such a simple formula so that we can easily present and analyze the tool's output. The intermediate formulas in the output correspond to the result of the individual reduction step described in Section 2.

```
Formula f3:
 NOT (|y PLUS x|  = |y|  + |x|)
Normalized formula f3:
 NOT (k0 = k1 + k2) AND FOR ALL e IN E. (m0(e) = y(e) + x(e)) AND
    (k0, k1, k2) = SUM {e in E,  TRUE } (m0(e), y(e), x(e))
Translated formula f3:
 NOT (k0 = k1 + k2) AND (k0, k1, k2) IN {(m0, y, x) | m0 = y + x }*
No more disjunctions:
  NOT (k0 = k1 + k2) AND k0 = u0  AND k1 = u1 AND k2 = u2 AND
   (u0, u1, u2) IN {(m0, y, x) | m0 = y + x }*
Semilinear set computation :
 ( m0, y, x )  | m0 = y + x,
semilinear set describing it is:
  List(0, 0, 0), List( List(1, 1, 0), List(1, 0, 1))
No more stars:
 NOT (k0 = k1 + k2) AND k0 = u0  AND k1 = u1 AND k2 = u2 AND
 u2 = 0 + 1*nu1 + 0 AND u1 = 0 + 0 + 1*nu0 AND u0 = 0 + 1*nu1 + 1*nu0
 AND ( NOT (mu0 = 0)  OR (nu1 = 0 AND nu0 = 0) )
---------------------
This formula is unsat
---------------------
```

The main problem we are facing for a more comprehensive evaluation of our tool is the lack of similar tools and benchmarks. Most benchmarks we were using are originally derived for reasoning about sets. Sometimes those formulas contain conditions that we do not need to consider when reasoning about multisets. This can especially be seen in Figure 3. Checking that an invariant on the size field of a data structure that implements a multiset is preserved after inserting 3 objects requires 0.4 seconds. Checking the same property for a data structure implementing a set requires 3.23 seconds.

We could also not compare the MUNCH tool with interactive theorem provers since our tool is completely automated and does not require any interaction.

In the future we plan to integrate our tool into theorem provers for expressive higher-order logics and to incorporate it into software verification systems.

Property	#set vars	#multiset vars	time (s)
Correctness of efficient emptiness check	1	0	0.40
Correctness of efficient emptiness check	0	1	0.40
Size invariant after inserting an element in a list	2	1	0.46
Size invariant after inserting an element in a list	0	2	0.40
Size invariant after deleting an element from a list	0	2	0.35
Allocating and inserting 3 objects into a container	5	0	3.23
Allocating and inserting 3 objects into a container	0	5	0.40
Allocating and inserting 4 objects into a container	6	0	8.35

Fig. 3. Measurement of running times for checking verification conditions that arise in proving correctness of container data structures. Please see tool web page for more details.

This will also enable us to obtain further sets of benchmarks. Our tool and the presented examples can be found at the following URL:
http://icwww.epfl.ch/~piskac/software/MUNCH/

References

1. Filliâtre, J.C., Marché, C.: The Why/Krakatoa/Caduceus platform for deductive program verification. In: CAV. pp. 173–177 (2007), http://www.lri.fr/~filliatr/ftp/publis/cav07.pdf
2. Ginsburg, S., Spanier, E.: Semigroups, Pressburger formulas and languages. Pacific Journal of Mathematics 16(2), 285–296 (1966)
3. Isabelle: Isabelle - a generic proof assistant. http://www.cl.cam.ac.uk/research/hvg/Isabelle/
4. KIV: KIV (Karlsruhe Interactive Verifier) http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/
5. Kuncak, V., Nguyen, H.H., Rinard, M.: Deciding Boolean Algebra with Presburger Arithmetic. J. of Automated Reasoning (2006), http://dx.doi.org/10.1007/s10817-006-9042-1
6. Lugiez, D.: Multitree automata that count. Theor. Comput. Sci. 333(1-2), 225–263 (2005)
7. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS. pp. 337–340 (2008), http://dx.doi.org/10.1007/978-3-540-78800-3_24
8. Piskac, R., Kuncak, V.: Decision procedures for multisets with cardinality constraints. In: VMCAI. No. 4905 in LNCS (2008)
9. Piskac, R., Kuncak, V.: Linear arithmetic with stars. In: CAV (2008)
10. Pottier, L.: Minimal solutions of linear diophantine systems: Bounds and algorithms. In: RTA. LNCS, vol. 488 (1991)
11. Pugh, W.: A practical algorithm for exact array dependence analysis. Commun. ACM 35(8), 102–114 (1992)