Automata-Based Representations for Arithmetic Constraints in Automated Verification *

Constantinos Bartzis and Tevfik Bultan

Department of Computer Science
University of California
Santa Barbara CA 93106, USA
{bar,bultan}@cs.ucsb.edu

Abstract. In this paper we discuss efficient symbolic representations for infinite-state systems specified using linear arithmetic constraints. We give new algorithms for constructing finite automata which represent integer sets that satisfy linear constraints. These automata can represent either signed or unsigned integers and have a lower number of states compared to other similar approaches. We experimentally compare different symbolic representations by using them to verify non-trivial specification examples. In many cases symbolic representations based on our construction algorithms outperform the polyhedral representation used in Omega Library, or the automata representation used in LASH.

1 Introduction

Symbolic representations enable verification of systems with large state spaces which cannot be analyzed using enumerative approaches. Recently, symbolic model checking has been applied to verification of infinite-state systems using symbolic representations that can encode infinite sets [8, 5, 7]. One class of infinite-state systems are systems that can be specified using linear arithmetic formulas on unbounded integer variables. Verification of such systems have many interesting applications such as monitor specifications, mutual exclusion protocols [5, 7], and parameterized cache coherence protocols [6]. In this paper we present new symbolic representations for linear arithmetic formulas and experimental results on efficiency of different symbolic representations.

There are two basic approaches to symbolic representation of linear arithmetic constraints in verification: 1) *Polyhedral representation*: In this approach linear arithmetic formulas are represented in a disjunctive form where each disjunct corresponds to a convex polyhedron. Each polyhedron corresponds to a conjunction of linear constraints [8, 7]. This approach can be extended to full Presburger arithmetic by including divisibility constraints (which can be represented as an equality constraint with an existentially quantified variable) [5, 2]. 2) *Automata representation*: An arithmetic constraint on v integer variables can be represented by a v-track automaton that accepts a string if it corresponds to a v-dimensional integer vector (in binary representation) that satisfies the corresponding arithmetic constraint [4, 11]. For both of these symbolic representations one can implement algorithms for intersection, union, complement, existential

* This work is supported in part by NSF grant CCR-9970976 and NSF CAREER award CCR-9984822.

quantifier elimination operations, and subsumption, emptiness and equivalence tests, and therefore use them in model checking.

In this paper we present new construction algorithms for the automata representation and also experimentally compare these different approaches. We give a new algorithm for constructing a finite automata representation for sets satisfying Presburger arithmetic formulas. The size of the resulting automaton in our construction has the same upper bound as the construction given in [4], however, our construction is also able to handle negative integers. The size of the resulting automaton in our construction is different than the construction given in [11]. We implemented our construction algorithm using the MONA tool [9] and integrated it to a set of tools for infinite-state model checking [12]. We experimented with a large set of examples. To compare the performance of our construction algorithm to other approaches we also integrated the LASH tool [1] which uses the automata construction given in [11], and Omega Library [2] which uses a polyhedral representation to the same set of tools and ran them on the same set of examples. Our experimental results show that our construction algorithm produces more compact representations than the construction algorithm given in [11]. Also automata representation is more efficient compared to the polyhedral representation used in [2].

2 Finite Automata Representation for Presburger Formulas

In this section we give a brief description of our algorithm for constructing a finite automaton that accepts the set of natural number tuples that satisfy a Presburger arithmetic formula on v variables. A full description and analysis of the algorithm and a comparison to other approaches is given in [3]. We encode numbers using their binary representation. A v-tuple of natural numbers $(n_1, n_2, ..., n_v)$ is encoded as a word over the alphabet $\{0,1\}^v$, where the i_{th} letter in the word is $(b_{i1}, b_{i2}, ..., b_{iv})$ and b_{ij} is the i_{th} least significant bit of number n_j.

The construction relies on a basic state machine (BSM) that performs linear arithmetic on non-negative integer variables. Each state of the BSM for $\sum_{i=1}^{v} a_i \cdot x_i$ is associated with a carry value. At any point, the BSM adds up the bit of the i_{th} variable of the current symbol a_i times for each i, plus the carry value of the current state. It writes the resulting bit to the output and moves to a new state according to the value of the new carry. The number of possible values of the carry (and thus the number of states) is $\sum_{i=1}^{v} |a_i|$.

A finite automaton (FA) for $\sum_{i=1}^{v} x_i = 0$ is similar to the BSM but has an extra sink state. Whenever the resulting bit is 1, the FA moves to the sink state, otherwise it continues as described before. The only initial and accepting state is the one associated with 0 carry. A FA for $\sum_{i=1}^{v} x_i < 0$ has no sink state. The accepting states are those associated with negative carries. The construction of FA for all other kinds of linear constraints $(\leq, >, \geq)$ is similar. If the right hand side of the equations or inequations is a non-zero constant c, the state with carry value of $-c$ will be the initial state of the FA. If no such state exists, we need to introduce more states corresponding to carry values between $-c$ and the carry value closest to $-c$. Thus the number of states now becomes at most: $S = |\min(-c, \sum_{a_i<0} a_i)| + |\max(-c, \sum_{a_i>0} a_i)|$. This is a tighter upper bound than the one given in [4], even though the construction algorithms are similar. An alternative way to cope with the constant term c is to stack $\log_2 c + 1$ BSMs, where the i_{th} BSM compares the resulting bit against the i_{th} bit of c. Now the total number of states is at most $(\log_2 c + 1) \cdot \sum_{i=1}^{v} |a_i|$. We can choose which alternative to use depending on the expected upper bound on the number of states. Finally, after constructing FA for atomic linear constraints (equations and inequations) we can construct FA for any Presburger formulas using standard automata operations such as intersection, union, complementation and projection.

Based on the same ideas we can construct FA for formulas on all integers (including negative), using 2's complement arithmetic. The procedure is based on the fact that in order for the FA to accept the encoding of a tuple of numbers, it must also accept the encoding of the same numbers with arbitrarily many sign bits (i.e. the most significant bit of each number repeated arbitrarily many times). The FA contains two clones of each state of the BSM, one accepting and one rejecting. For equations, looping transitions in the BSM that write 0 go to the according accepting clone. All other transitions that write 0 go to the rejecting clone. All transitions that write 1 in the BSM go to the sink state. For inequations, looping transitions that write 1 go to the accepting clone and those which write 0 go to the rejecting clone. Any other transition goes to the appropriate accepting clone, iff by repeatedly receiving the same combination of bits the BSM will eventually enter a loop which writes 1. Otherwise it goes to the rejecting clone. These FA are only twice as large as those for non-negative integers described before.

Note that our construction algorithm and the one in [11] result in different automata, because the accepted languages are different (one is the reverse of the other). Moreover, we can prove the same or lower upper bound on the size of the resulting automata. Finally, in our algorithm, once a state has been created, all transitions originating from it can be computed immediately (as opposed to [11]), which is more convenient when transitions are stored using BDDs.

3 Implementation and Experiments

In [10] polyhedral and automata representation for arithmetic constraints are compared experimentally for reachability analysis of several concurrent systems. The results show no clear winner. On some problem instances the polyhedral representation is superior, on some others automata representation is. Our experimental setup is more reliable compared to [10]. In [10] boolean variables are mapped to integer variables when polyhedral representation is used. This is an inefficient encoding which gives an unfair advantage to the automata representation. In our experiments boolean variables are not mapped to integers in any representation. Also, our tools perform full CTL model checking including liveness properties instead of just reachability analysis discussed in [10].

We integrated our construction algorithms to an infinite state CTL model checker built on top of the Composite Symbolic Library [12]. The Composite Symbolic Library defines an abstract interface for the operations used in symbolic verification [12]. To integrate a new symbolic representation to the Composite Symbolic Library one implements this abstract interface with specialized operations. Composite Symbolic Library supports a disjunctive composite representation for formulas on integer and boolean variables. A disjunctive composite representation is in the form $\bigvee_{i=1}^{n} \bigwedge_{t \in T} p_{it}$ where p_{it} denotes the formula of type

t (which could be integer or boolean) in the ith disjunct, and n and T denote the number of disjuncts and the set of variable types ($T = \{integer, boolean\}$)), respectively. The methods such as intersection, union, complement, satisfiability check, subsumption test, which manipulate composite representations in the above form are implemented in the Composite Symbolic Library by calling the operations on integer and boolean formula representations [12].

We integrated five different symbolic representations to the Composite Symbolic Library. The first three use the disjunctive composite representation described above to combine formulas on integer and boolean variables. We used the BDD representation for boolean formulas. We implemented three different integer formula representations using LASH [1]) (version V3), Omega [2] (version V2), and our automata construction algorithm (version V1) which uses MONA automata package [9] as an automata manipulator. We also implemented two automata based representations using LASH (version V5) and our construction algorithms (version V4) again built on top of MONA automata package, for both boolean and integer variables without using the disjunctive composite representation. The states of both boolean and integer variables can be represented in an automaton, hence one can avoid using the disjunctive composite representation.

We experimented with a large set of examples. Specification of the examples and properties are available at: http://www.cs.ucsb.edu/~bultan/composite/.

The results of our experimental evaluation of different representations for linear integer arithmetic constraints are shown in Table 1. We obtained the experimental results on a SUN ULTRA 10 work station with 768 Mbytes of memory, running SunOs 5.7. For each version of the verifier we recorded the following statistics: 1) Time elapsed during the construction of the symbolic representation of the transition system, shown in the table as CT. 2) Time elapsed during the verification process, shown as VT. It includes the time needed for forward or backward fixpoint computations, however, it excludes the construction time (CT). 3) The maximum amount of memory used by the verifier, shown as Mem. Also for V1, V3, V4 and V5 that use automata as a symbolic representation we recorded the size (number of states) of the automaton representing the transition system, shown as TRS, and the size of the largest automaton computed during the fixpoint computation, shown as MS. As discussed above our automata construction algorithm used in versions V1 and V4 uses MONA automata package. MONA automata package uses BDDs to store the transition relation of the automata. Therefore, to make the comparison with LASH fair, instead of giving the number of automaton states for versions V1 and V4, we give the total number of BDD nodes used in the MONA representation. For the barber and ticket problems we used forward fixpoint computation, while for all other problems backward fixpoint computations were used.

By carefully inspecting Table 1 one can make the following remarks. Versions V4 and V5 that use only automata as a single representation for both integer and boolean variables require less memory, both for the transition relation and the fixpoint iterates, than V1, V2 and V3 that use disjunctive representations. This can be explained by the fact that deterministic minimal automata are canonical, while the composite symbolic representation is not. For most of the examples but not all, V4 performs better than V5 both in construction and verification times and memory requirements. This is in accordance to the measured sizes of the two automata implementations. On the other hand, by inspecting the data for

Table 1. Experimental results for performance of different symbolic representations for integer arithmetic constraints. Time measurements appear in seconds and memory measurements in Mbytes. For each problem instance the following values are shown: time elapsed during the construction of the symbolic representation of the transition system (CT), time elapsed during verification process (VT), the maximum amount of memory used by the verifier (Mem). For the automata representations we also give the number of states in the automata representing the transition relation (TRS) and the number of states in the automata representing the fixpoint iterate with the maximum size (MS). The instances marked ↑ ran out of memory or did not converge in 5000 seconds.

Problem Instance	Disjunctive Composite Representation																						
	V1 - Automata					V2 - Omega			V3 - LASH					V4 - Automata					V5 - LASH				
	CT	VT	Mem	TRS	MS	CT	VT	Mem	CT	VT	Mem	TRS	MS	CT	VT	Mem	TRS	MS	CT	VT	Mem	TRS	MS
BARBERM2-1	0.13	0.07	8	184	26	0.22	0.11	7	0.31	0.23	6	198	25	0.05	0.01	0.45	115	40	1.71	0.22	0.36	276	46
BARBERM3-1	0.14	0.08	9	241	26	0.2	0.11	7	0.39	0.29	6	257	25	0.07	0.01	0.46	124	42	2.39	0.26	0.4	311	52
BARBERM4-1	0.15	0.09	9	298	26	0.21	0.13	7	0.44	0.36	6	316	25	0.07	0.02	0.5	133	44	3.1	0.28	0.43	346	58
BARBERM2-2	0.12	0.08	8	184	26	0.18	0.11	7	0.31	0.25	6	198	25	0.05	0.01	0.45	115	40	1.72	0.22	0.36	276	46
BARBERM3-2	0.13	0.09	9	241	26	0.19	0.12	7	0.37	0.31	6	257	25	0.06	0.02	0.46	124	42	2.3	0.26	0.4	311	52
BARBERM4-2	0.15	0.1	9	298	26	0.21	0.13	7	0.44	0.36	6	316	25	0.07	0.02	0.5	133	44	3.09	0.29	0.43	346	58
BARBERM2-3	0.12	0.08	8	184	26	0.19	0.11	7	0.31	0.25	6	198	25	0.06	0.01	0.45	115	40	1.71	0.22	0.36	276	46
BARBERM3-3	0.14	0.08	9	241	26	0.19	0.12	7	0.37	0.31	6	257	25	0.06	0.02	0.46	124	42	2.39	0.26	0.4	311	52
BARBERM4-3	0.14	0.1	10	298	26	0.22	0.13	7	0.45	0.37	6	316	25	0.07	0.02	0.5	133	44	3.1	0.28	0.04	346	58
BARBERMP-1	0.2	0.22	9	1266	131	0.23	0.33	7	1.01	1.45	6	1282	139	0.2	0.12	0.7	1103	131	7.92	1.19	2.7	1544	156
BARBERMP-2	0.21	0.21	9	1266	131	0.22	0.34	7	1.01	1.46	6	1282	139	0.2	0.12	0.7	1103	131	7.93	1.21	0.27	1544	156
BARBERMP-3	0.2	0.22	9	1266	131	0.22	0.33	7	1.01	1.45	6	1282	139	0.19	0.12	0.7	1103	131	7.91	1.22	2.7	1544	156
BAKERY2-1	0.1	0.04	8	80	31	0.13	0.11	7	0.18	0.24	6	96	42	0.06	0.02	0.38	117	36	1.56	0.27	0.27	236	73
BAKERY2-2	0.11	0.08	9	80	21	0.13	0.16	7	0.18	0.47	6	96	25	0.05	0.04	0.42	117	28	1.53	0.51	0.26	236	55
BAKERY3-1	0.17	1.52	33	558	306	0.3	3.92	10	0.64	10.39	6	569	424	0.37	0.77	0.64	646	382	6.22	6.5	1.83	1247	588
BAKERY3-2	0.16	3.6	100	558	305	0.3	13.1	20	0.64	23.78	7	569	340	0.36	1.64	0.79	646	117	6.14	13.61	1.76	1247	230
BAKERY4-1	0.49	38.21	420	3284	2437	1.09	142.96	43	2.61	340.02	10	2691	3321	12.14	60.5	1.82	3444	6010	63.48	273.01	30.38	5542	5667
BAKERY4-2	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	12.15	113	1.74	3444	869	63.45	546.81	29.75	5542	1027
TICKET2-1	0.1	0.18	10	168	117	0.13	0.22	7	0.26	0.57	6	188	116	0.07	0.1	0.56	179	132	2	1.04	0.43	263	161
TICKET2-2	0.11	0.18	10	168	117	0.13	0.22	7	0.27	1.15	6	188	116	0.06	0.11	0.56	179	132	1.99	1.05	0.43	263	161
TICKET3-1	0.13	0.91	33	315	107	0.17	1.99	10	0.46	4.51	7	354	627	0.16	1.36	0.93	333	562	4.78	8.93	1.93	492	839
TICKET3-2	0.14	1.67	33	315	107	0.18	1.99	13	0.45	12.34	8	354	627	0.16	1.3	0.93	333	562	4.78	8.93	1.93	492	839
TICKET4-1	0.16	17.24	221	504	2775	0.23	27	27	0.7	116.97	8	568	2961	0.66	14.6	2.09	531	2336	11.86	74.72	10.2	789	4103
TICKET4-2	0.16	17.29	221	504	2775	0.23	26.94	27	0.71	116.79	17	568	2961	0.66	14.6	2.09	531	2336	11.86	74.74	10.2	789	4103
COHERENCE-1	0.19	0.25	10	1150	102	0.26	0.28	7	1.05	5.92	6	1203	263	0.11	0.33	0.7	730	86	3.56	4.51	1.52	1377	237
COHERENCE-2	0.2	0.52	15	1150	51	0.27	1.99	10	1.04	8.49	7	1203	568	0.11	0.51	0.74	730	161	3.57	5.78	1.67	1377	472
COHERENCE-3	0.21	2.46	30	1150	1013	0.26	7.3	14	1.02	47.16	7	1203	2019	0.11	2.2	0.98	730	544	3.55	25.27	3.8	1377	1216
COHERENCE-4	0.19	22.43	334	1150	448	0.27	42.91	39	1.04	206.13	12	1203	746	0.11	31.4	2	730	239	3.57	196.63	2.58	1377	641
PC5	0.13	0.03	8	440	54	0.15	0.05	7	0.43	0.92	6	587	74	0.09	0.04	0.55	244	61	4.26	0.61	0.94	680	144
PC10	0.19	0.06	8	700	54	0.22	0.08	7	0.62	1.55	7	917	74	0.19	0.06	0.79	283	61	11.9	0.7	1.26	811	166
RW4	0.09	0.01	7	50	3	0.12	0.01	6	0.12	0.01	6	40	3	0.02	0.01	0.03	71	5	0.63	0.03	0.17	136	18
RW8	0.13	0.01	8	100	3	0.18	0.01	7	0.2	0.02	6	80	3	0.04	0.01	0.31	143	5	2.61	0.05	0.24	256	30
RW16	0.23	0.02	10	200	3	0.36	0.02	7	0.42	0.05	6	160	3	0.12	0.01	0.04	287	5	13.01	0.09	0.41	496	54
RW32	0.65	0.03	19	400	3	1	0.03	8	1.2	0.09	7	320	3	0.38	0.02	0.07	575	5	76.28	0.17	0.76	976	102
RW64	2.83	0.06	49	800	3	3.86	0.06	13	4.61	0.18	10	640	3	1.82	0.07	1.33	1151	5	536.96	0.35	1.77	1936	198
SIS	694.87	0.34	19	107873	422	2.31	0.06	12	3.49	0.04	7	192	19	20.7	0.69	1.26	54568	431	152.85	1.92	39.22	95617	850
LIGHTCONTROL	0.2	0.08	10	108	12	0.18	0.07	7	0.28	0.2	6	72	12	0.07	0.02	0.04	187	27	2.41	0.3	0.34	301	70
INSERTIONSORT	0.11	0.01	7	140	13	0.11	0.15	7	0.21	0.67	6	173	86	0.06	0.02	0.46	127	19	1.5	0.77	0.43	231	101

V1, V2, V3, we can see a clear manifestation of time-memory tradeoff. For most examples V1 is faster than V2 and V3, but uses more memory. V3 consumes less memory, but is usually slower than the other two. V2 appears to be a good compromise between time and memory efficiency. Now, when comparing the composite versions V1 and V3 with their "pure automata" counterparts V4 and V5 respectively, we can conclude that the later generally perform better than the earlier, with the exception of most construction time measurements for V5 being greater that those for V3.

Based on these results we conclude that the versions based on our automata construction algorithms for linear integer arithmetic formulas and implemented using MONA automata package (V1 and V4) are the most time efficient of all, with V4 being more memory efficient than V1. If a composite representation must be used, Omega library based V2 can be a good time-memory compromise. Finally LASH based V5 is a close competitor to V4, but suffers from high construction times.

References

1. The Liège Automata-based Symbolic Handler (LASH). Available at `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.
2. The Omega project. `http://www.cs.umd.edu/projects/omega/`.
3. C. Bartzis and T. Bultan. Efficient symbolic representations for arithmetic constraints in verification. Technical Report TRCS-2002-16, Computer Science Department, University of California, Santa Barbara, June 2002.
4. A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In H. Kirchner, editor, *Proceedings of the 21st International Colloquium on Trees in Algebra and Programming - CAAP'96*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer-Verlag, April 1996.
5. Tevfik Bultan, Richard Gerber, and William Pugh. Model-checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results. *ACM Transactions on Programming Languages and Systems*, 21(4):747–789, 1999.
6. G. Delzanno and T. Bultan. Constraint-based verification of client-server protocols. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, 2001.
7. Giorgio Delzanno and Andreas Podelski. Constraint-based deductive model checking. *Journal of Software Tools and Technology Transfer*, 3(3):250–270, 2001.
8. T. A. Henzinger, P. Ho, and H. Wong-Toi. Hytech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
9. Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual*. BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus, January 2001.
10. T. R. Shiple, J. H. Kukula, and R. K. Ranjan. A comparison of Presburger engines for EFSM reachability. In *Proceedings of the 10th International Conference on Computer-Aided Verification*, 1998.
11. P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In S. Graf and M. Schwartzbach, editors, *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 1–19. Springer, April 2000.
12. T. Yavuz-Kahveci, M. Tuncer, and T. Bultan. Composite symbolic library. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 335–344. Springer-Verlag, April 2001.