Construction of efficient BDDs for Bounded Arithmetic Constraints *

Constantinos Bartzis and Tevfik Bultan

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

Abstract. Most symbolic model checkers use BDDs to represent arithmetic constraints over bounded integer variables. The size of such BDDs can be exponential on the number and size (in bits) of the integer variables in the worst case. In this paper we show how to construct linear-sized BDDs for linear integer arithmetic constraints. We present basic constructions for atomic equality and inequality constraints and extend them to handle arbitrary linear arithmetic formulas. We also present three alternative ways of handling out-of-bounds transitions, and discuss multiple bounds on integer variables. We experimentally compare our approach to other BDD-based symbolic model checkers and demonstrate that the algorithms presented in this paper can be used to improve their performance significantly.

1 Introduction

Performance of a symbolic model checker depends on the efficiency of the algorithms for the BDD construction and the sizes of the generated BDD representations. In this paper we address both these issues for linear arithmetic constraints on bounded integer variables. BDD-based model checkers represent bounded integer variables by mapping them to a set of Boolean variables using a binary encoding. Our experiments show that the state of the art BDD-based model checkers use inefficient algorithms for BDD construction from linear arithmetic constraints and fail to generate compact BDD representations for them [11, 8, 1]. Handling linear arithmetic constraints efficiently is an important problem since such constraints are common in reactive system specifications. For example, the distribution files for the BDD-based model checker NuSMV contain specifications with linear arithmetic constraints, however, the verification time for these specifications for NuSMV does not scale when the bounds on integer variables are increased. The algorithms and complexity results presented in this paper demonstrate that this inefficiency is not inherent to the BDD data structure and can be avoided.

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

We present algorithms for constructing efficient BDD representations from atomic arithmetic constraints of the form $\sum_{i=1}^{v} a_i \cdot x_i \ \# \ a_0$, where $\# \in \{=, \neq, >, \geq, \leq, <\}$. We show that the size of the resulting BDD is linear in the number of variables and the number of bits used to encode each variable. We also show that the time complexity of the construction algorithm is the same. We also give bounds for BDDs for linear arithmetic formulas which can be obtained by combining atomic arithmetic constraints with boolean connectives. We show that the resulting BDDs for linear arithmetic formulas are still linear in the number of variables and the number of bits used to encode each variable. We extend the construction algorithms to handle transitions which can take the bounded integer variables out-of-bounds. We present three different approaches for handling out-of-bounds transitions and show that all of them preserve the complexity results. We also generalize the construction algorithms to multiple bounds on integer variables. We show that as long as all the bounds are powers of two the complexity results are preserved. One interesting result is that multiple bounds which are not powers of two cause the BDD size to be exponential in the number of variables in the worst case.

The problem of inefficient BDD representation of arithmetic constraints in symbolic model checkers has been pointed out in [7, 12]. In [7], the problem for SMV is handled by writing a preprocessor and fixing the BDD variable order. However, as we show in this paper, this extra step is not necessary since efficient BDDs can be directly constructed from a set of linear arithmetic constraints. In [12], the problem is solved only for constraints of the form $x + y = z$, where x, y and z can be variables or constants. Even though such constraints arise very often in practice, our algorithms are more general without sacrificing efficiency. Binary Moment Diagrams (BMDs) [4] and Hybrid Decision Diagrams (HDDs) [9] are data structures designed to represent arithmetic expressions and handle arithmetic operations on word-level verification where an array of binary bits can be referred as an integer. These data structures can also be used to construct linear-sized BDDs from linear arithmetic constraints. However, in this paper, we show that one can construct linear-sized BDDs from linear arithmetic constraints directly, without using these data structures. Hence, the algorithms we present can be easily integrated to a BDD-based model checker.

2 Atomic equality constraints

Given a set of v integer variables $x_i, 1 \leq i \leq v$ such that $0 \leq x_i < 2^b$ and a linear equation of the form $\sum_{i=1}^{v} a_i \cdot x_i = a_0$ we construct a BDD with $v \cdot b$ boolean variables $x_{i,j}, 1 \leq i \leq v, 0 \leq j < b$ which evaluates to $\mathbf{1}$ iff $\sum_{i=1}^{v} a_i \cdot (\sum_{j=0}^{b-1} x_{i,j} \cdot 2^j) = a_0$. In other words the BDD variables $x_{i,j}$ represent the binary digits of the integer variables and the BDD evaluates to $\mathbf{1}$ iff the equation is satisfied by the valuation $x_i = \sum_{j=0}^{b-1} x_{i,j} \cdot 2^j$ for $1 \leq i \leq v$. We show that such a BDD has $O(v \cdot b \cdot \sum_{i=1}^{v} |a_i|)$ nodes, i.e. the size of the BDD is linear on the number of boolean variables. Note that in general the size of a BDD can be exponential on

BDD construction algorithm for equations $\sum_{i=1}^{v} a_i \cdot x_i = a_0$ *on b-bit variables*

```
Procedure: node(i, j, c)
//Constructs a node in level i of layer j with label c
BDDnode n
    n.index := j · v + i
    if (i = v and j = b − 1)
        if (c = 0) n.low := 1
        else n.low := 0
        if (c + a_v = 0) n.high := 1
        else n.high := 0
    else if (i = v)
        if (c is even) n.low := node(1, j + 1, c/2)
        else n.low := 0
        if (c + a_v is even) n.high := node(1, j + 1, (c + a_v)/2)
        else n.high = 0
    else
        n.low := node(i + 1, j, c)
        n.high = node(i + 1, j, c + a_v)
    return n

Main: return node(1, 0, −a_0)
```

Fig. 1. BDD construction algorithm for equations

the number of boolean variables and experimental results show that state of the art model checkers produce exponentially large BDDs.

The algorithm is given in Figure 1. The constructed BDD consists of b layers of v levels each. The j_{th} layer corresponds to the j_{th} least significant bit of each integer variable and the i_{th} level in a layer corresponds to the i_{th} integer variable. Every node in a level is labeled with an integer c between $-\sum_{i=0}^{v}|a_i|$ and $\sum_{i=0}^{v}|a_i|$. In particular, the label of a node in the 1_{st} level of the j_{th} layer corresponds to a value of the carry c resulting from the computation of $\sum_{i=1}^{v} a_i \cdot (\sum_{n=0}^{j-1} x_{i,j} \cdot 2^n) - a_0$, where $x_{i,j}$s are the values of the BDD variables along one of the paths from the root to that node. Furthermore, the label of a node in the k_{th} level, $2 \leq k \leq v$, of the j_{th} layer is the value $c + \sum_{i=1}^{k-1} a_i \cdot x_{i,j}$, where $x_{i,j}$s are the values of the BDD variables along one of the paths from the node in the 1_{st} level of the j_{th} layer with label c to that node.

As an example consider the linear equation $2x - 3y = 1$, where x and y are 4 bits long. Figure 2 shows the structure of a complete intermediate layer (inside the dashed rectangle) of the corresponding BDD. The nodes outside the rectangle comprise the first level of the next layer. The complete BDD is shown in Figure 3. For all Figures, edges not shown point to **0** terminal node. Note that the BDDs constructed by the algorithms in this paper are not necessarily reduced. Standard BDD reduction needs to be applied after the construction.

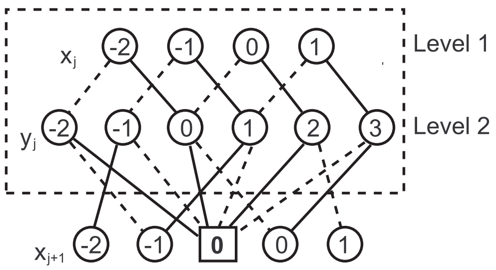

Fig. 2. Layer of a BDD for $2x - 3y = 1$

Theorem 1. *The algorithm given in Figure 1 constructs a BDD representing the linear equation* $\sum_{i=1}^{v} a_i \cdot x_i = a_0$ *on b-bit non-negative integer variables. The time complexity of the algorithm and the size of the resulting BDD is* $O(v \cdot b \cdot \sum_{i=1}^{v} |a_i|)$.

Proof. For the purposes of the proof we can think of a BDD as a bit-serial processor as described in [3]. Such a processor computes a Boolean function by examining the arguments x_1, x_2, and so on in order, producing output 0 or 1 after the last bit has been read. It requires internal storage to store enough information about the arguments it has already seen to correctly deduce the value of the function from the values of the remaining arguments. Trivially it can store all the values of the arguments it has already seen by using exponentially large storage. In our case we can show that linear storage is needed. The size of the storage consumed by the processor translates to the number of nodes in the BDD.

The ordering of the boolean variables $x_{i,j}$ is lexicographical primarily on j and secondarily on i or equivalently the index of variable $x_{i,j}$ is $j \cdot v + i$. The *index* of the root is 1. One can easily verify that any internal node with index *index* points to a node with index $index + 1$, except for the nodes with index $b \cdot v$ that point to the terminal nodes. Thus the constructed BDD is consistent with the ordering mentioned above. Therefore the bit-serial processor equivalent of the BDD first processes the least significant bit of the integer variables $x_1, x_2, ..., x_v$ in this order, then it processes the second least significant bits and so on. In the end the processor needs to verify whether or not $\sum_{i=1}^{v} a_i \cdot x_i = a_0$ or equivalently $\sum_{i=1}^{v} a_i \cdot (\sum_{j=0}^{b-1} x_{i,j} \cdot 2^j) = a_0$ or

$$-a_0 + \sum_{j=0}^{b-1} 2^j \cdot (\sum_{i=1}^{v} a_i \cdot x_{i,j}) = 0 \tag{1}$$

To accomplish this the processor gradually computes the left hand side of equation (1) bit by bit and compares it against zero. If at any point the comparison fails it immediately evaluates to **0**, otherwise it continues. It starts with an initial value of $-a_0$ and then gradually adds to it $a_i \cdot x_{i0}$ as it reads the values of $x_{1,0}$ up to $x_{v,0}$ and stores the intermediate result $-a_0 + \sum_{i=1}^{l} a_i \cdot x_{i,0}$ every time. Note

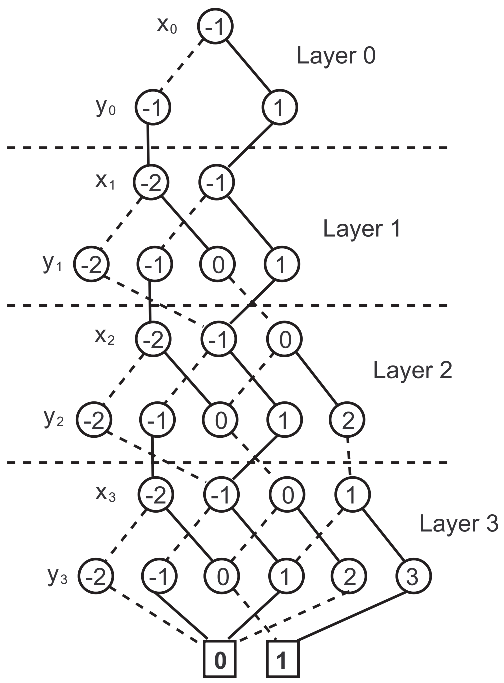

Fig. 3. BDD for $2x - 3y = 1$ for 4-bit variables

that the value stored is shown as the label of each BDD node in the Figures. At the end of processing layer 0, if the result is an odd number (or the resulting bit is 1) the processor immediately evaluates to **0**, since what remains to add, namely $\sum_{j=1}^{b-1} 2^j \cdot (\sum_{i=1}^{v} a_i \cdot x_{i,j})$, is an even number and there is no chance that the final result will be zero. Otherwise, the intermediate result at this point divided by two is equal to the remaining carry c, i.e. $c = (-a_0 + \sum_{i=1}^{v} a_i \cdot x_{i,0})/2$. The value of the carry is the only piece of information that needs to be stored at this point. If we divide both sides of equation (1) by 2 we will get:

$$c + \sum_{j=1}^{b-1} 2^{j-1} \cdot (\sum_{i=1}^{v} a_i \cdot x_{i,j}) = 0 \qquad (2)$$

Now the processor needs to verify equation (2) and this task is similar to the initial one, so the processor continues to operate in a similar manner. In the end, in order for the final result to be 0 the final carry has to be also 0. In that case the processor evaluates to **1**, otherwise it evaluates to **0**. This concludes the proof of correctness of our construction algorithm.

For the proof of termination and complexity, the fundamental question that needs to be answered is how many different intermediate results need to be stored at any point during the operation of the bit-serial processor or in other words how many BDD nodes are there at any level. The number of nodes at any level

BDD construction algorithm for inequations $\sum_{i=1}^{v} a_i \cdot x_i < a_0$ *on b-bit variables*

```
Procedure: node(i, j, c)
//Constructs a node in level i of layer j with label c
BDDnode n
    n.index := j · v + i
    if (i = v and j = b − 1)
        if (c < 0) n.low := 1
        else n.low := 0
        if (c + a_v < 0) n.high := 1
        else n.high := 0
    else if (i = v)
        if (c is even) n.low := node(1, j + 1, c/2)
        else n.low := node(1, j + 1, (c − 1)/2
        if (c + a_v is even) n.high := node(1, j + 1, (c + a_v)/2)
        else n.high = node(1, j + 1, (c + a_v − 1)/2)
    else
        n.low := node(i + 1, j, c)
        n.high = node(i + 1, j, c + a_v)
    return n

Main: return node(1, 0, −a_0)
```

Fig. 4. BDD construction algorithm for inequations

is bounded by the size of the range defined by the least and the greatest label in that level. If the labels of the nodes at level $j \cdot v + 1$ belong to a range of size $n_{1,j}$, then level $j \cdot v + 2$ has at most $n_{2,j} = n_{1,j} + |a_1|$ nodes, level $j \cdot v + 3$ has at most $n_{3,j} = n_{1,j} + |a_1| + |a_2|$ nodes and so on. Finally level $j \cdot v + v + 1 = (j+1) \cdot v + 1$ has at most $n_{1,j+1} = (n_{1,j} + \sum_{i=1}^{v} |a_i|)/2$ nodes because that many are the different values of the carry that need to be stored, as described earlier. Initially $n_{1,0} = 1$ and by induction one can prove that no $n_{i,j}$ is larger than $2 \cdot \sum_{i=1}^{v} |a_i|$. There are $v \cdot b$ levels in the BDD so the total number of nodes is $O(v \cdot b \cdot \sum_{i=1}^{v} |a_i|)$, i.e. the size of the constructed BDD is linear on both v and b. Each node is created once if we store each of them in a hash table indexed by i, j and c and the creation of a node requires a fixed amount of work, so the complexity of our algorithm is $O(v \cdot b \cdot \sum_{i=1}^{v} |a_i|)$. Note that a tighter bound for the BDD size is $O(b \cdot \sum_{i=1}^{v} |a_i| \cdot (1 + v - i))$. Interestingly this indicates that the size of the BDD is minimized if the integer variables are ordered in increasing order of the absolute values of their coefficients.

3 Atomic inequality constraints

Next we show how to construct BDDs for inequations of the form $\sum_{i=1}^{v} a_i \cdot x_i < a_0$, $0 \leq x_i < 2^b$. Note that we can transform all other kinds of linear inequations

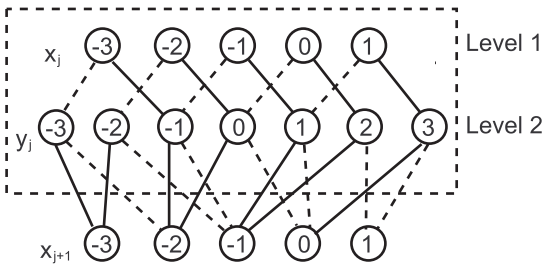

Fig. 5. Layer of a BDD for $2x - 3y < 1$

$(\leq, >, \geq)$ to this form by changing the signs of the coefficients and/or adding 1 to the constant term a_0. The algorithm is similar to the one for equations and is shown in Figure 4. There are only two differences. First, after having processed an equal number of bits from all integer variables (lines 9-13 of the algorithm) we do not require the resulting bit to be 0. The bit-serial processor only computes the correct value of the remaining carry and proceeds to the next level. Second, in order for the inequality to hold after all bits have been processed (lines 4-8 of the algorithm), the remaining carry has to be negative. Obviously, these two modifications do not change the bound on the number of nodes in the BDD, which is again $O(v \cdot b \cdot \sum_{i=1}^{v} |a_i|)$. This proves the following theorem.

Theorem 2. *The algorithm given in Figure 4 constructs a BDD representing the linear inequation $\sum_{i=1}^{v} a_i \cdot x_i < a_0$ on b-bit non-negative integer variables. The time complexity of the algorithm and the size of the resulting BDD is $O(v \cdot b \cdot \sum_{i=1}^{v} |a_i|)$.*

As an example consider the linear inequation $2x - 3y < 1$, where x and y are 4 bits long. Figures 5 and 6 show the structure of an intermediate layer and the complete BDD before being reduced.

4 Linear arithmetic formulas

In symbolic model checking BDDs are subjected to operations such as intersection, union, negation, etc. as well as subsumption and equivalence tests. The time and space complexity of these operations depends on the size of the operands. The complexity of negation is $O(1)$, as it involves only swapping the terminal nodes **0** and **1**, but the complexity of intersection and union which are very frequently used operations is $O(n_1 \cdot n_2)$, where n_1 and n_2 are the sizes of the operands. Suppose that one performs one of these operations on two BDDs representing the constraints $\sum_{i=1}^{v} a_i \cdot x_i = a_0$ and $\sum_{i=1}^{v} b_i \cdot x_i = b_0$ whose sizes are $O(v \cdot b \cdot \sum_{i=1}^{v} |a_i|)$ and $O(v \cdot b \cdot \sum_{i=1}^{v} |b_i|)$ respectively, as proved earlier. One would expect the size of the resulting BDD to be $O(v^2 \cdot b^2 \cdot \sum_{i=1}^{v} |a_i| \cdot \sum_{i=1}^{v} |b_i|)$. Actually this is a pessimistic estimation. The resulting BDD will have again $v \cdot b$ layers, corresponding to a bit-serial processor that examines each of the $x_{i,j}s$ one by one

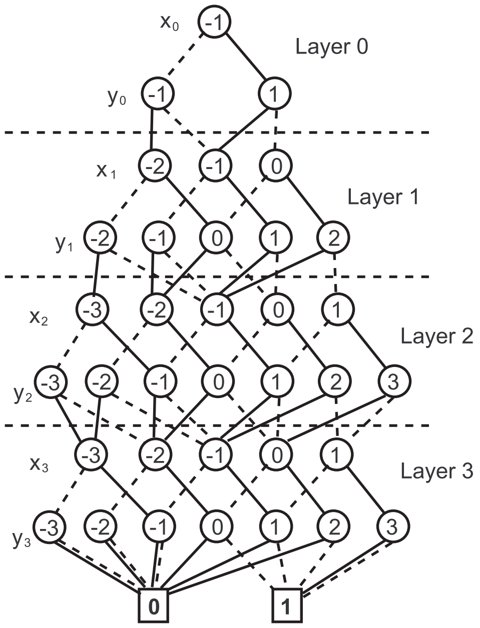

Fig. 6. BDD for $2x - 3y < 1$ for 4-bit variables

as before. The only difference is that now it needs to remember the intermediate results from both BDDs and thus every layer will have $O(\sum_{i=1}^{v}|a_i| \cdot \sum_{i=1}^{v}|b_i|)$ nodes and there will be $O(v \cdot b \cdot \sum_{i=1}^{v}|a_i| \cdot \sum_{i=1}^{v}|b_i|)$ nodes in total. Clearly the same argument holds for more than two linear constraints, which proves the following Theorem.

Theorem 3. *Given a linear arithmetic formula on b-bit non-negative integer variables consisting of n atomic constraints of the form* $\sum_{i=1}^{v} a_{i,j} \cdot x_i = a_{0,j}$, $1 \le j \le n$ *and boolean connectives* $\neg, \wedge, \vee$, *one can construct a BDD of size* $O(v \cdot b \cdot \prod_{j=1}^{n}\sum_{i=1}^{v}|a_{i,j}|)$ *representing the formula in time* $O(v \cdot b \cdot \prod_{j=1}^{n}\sum_{i=1}^{v}|a_{i,j}|)$.

The conclusion is that when basic operations are performed by a model checker on BDDs representing linear arithmetic constraints on bounded integers, the size of intermediate BDD representations remains linear on the number and size of the integer variables, i.e. the space and time complexity of operations does not blow up with respect to these two parameters. This is very important since such "blow ups" are a common drawback of BDD based model checking.

Note that since satisfiability checking for BDDs can be performed in constant time, one can decide the satisfiability of n linear constraints on v b-bit integer variables in time $O(v \cdot b \cdot \prod_{j=1}^{n}\sum_{i=1}^{v}|a_{i,j}|)$, using our construction algorithm. Hence, the complexity is linear in v and b, and only exponential in the number of atomic constraints n. This problem is NP-complete even if $b = 1$ [10], which implies that there is no algorithm which is polynomial in v, b and n, unless P=NP.

5 Handling multiplication

An inherently unavoidable shortcoming of BDDs is their inability to efficiently represent arithmetic constraints involving multiplication between variables. In [3] it has been proven that the size of such BDDs has a lower bound exponential on b, the size of the integer variables, regardless of the variable ordering. The good news is that by choosing the variable ordering we defined earlier and by slightly modifying our construction algorithm one can accommodate multiplication and keep the size of the produced BDD exponential only in b and the number of integer variables involved in multiplications which is in many cases less than v, the total number of integer variables. The idea supporting this argument is the following. Suppose we want to construct a BDD for an arithmetic formula on $v = m + l$ integer variables, in which only m variables are multiplied with other variables in the formula (which we will call m-variables) and the rest l variables (which we will call l-variables) are only multiplied with constants, forming the "linear part" of the formula. In the worst case, the bit-serial processor equivalent of the BDD will need to remember the exact values of the m variables and the intermediate results c of the computation of the "linear part", as described earlier. The number of levels remains the same $v \cdot b$. At any level, when an m-variable is processed all nodes are doubled in the next level, thus remembering the new bit of the m-variable and the various cs are propagated properly. When an l-variable is processed, the processor behaves exactly as in the linear case. The number of nodes is doubled $m \cdot b$ times and consequently the size of the BDD will be $O(2^{m \cdot b} \cdot l \cdot b \cdot \sum_{i=1}^{v} |a_i|)$. Of course this is not an impressive result but nevertheless indicates a complexity that is exponentially dependent on m and not v. In many practical cases if m is non-zero it is at least much less than v. Note that by choosing a different variable ordering one can end up with BDDs of exponential size in both b and v.

6 Handling overflows

When constructing BDDs to represent the transition relation of a system, special care is needed in order to handle possible overflows. For example consider a transition labeled by $x' = x + 1$, where x is the current state variable and x' is the next state variable. They are both 2-bit non-negative integers ranging between 0 and 3. When $x = 3$ and the transition is taken, what is an appropriate value of x in the next state, since it cannot be 4? We consider three alternatives:

1. The transition is not taken and the next state is empty.
2. Modular arithmetic is performed and $x = 0$ in the next state.
3. An "Out of bounds" error is detected and reported.

BDD construction for the transition relation depends on the choice of one of these three alternative approaches. For our example, an intermediate layer and the complete BDDs for all three approaches are shown in Figure 7. The construction algorithm described earlier follows the first approach. The difference between the

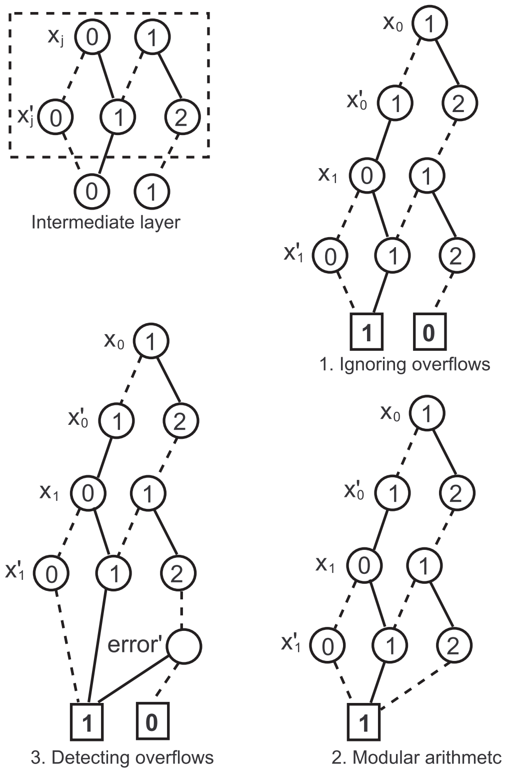

Fig. 7. Alternative ways to handle overflows

three approaches is in the edges generated by lines 6 and 8 of the construction algorithm, which correspond to the case where the b least significant bits of the variables (in our example $b = 2$) satisfy the equation but there is a remaining non-zero carry indicating an overflow. The first approach points all such edges to the **0** terminal node, thus making the BDD to evaluate to **0** whenever an overflow occurs. The second approach points all such edges to **1**, thus ignoring overflows and performing modular arithmetic. The third approach is a bit more involved. There is an extra global boolean variable *error* in the end of the variable ordering. All edges generated by lines 6 and 8 of the construction algorithm point to a node with the *index* of $error'$ and $low = \mathbf{0}$ and $high = \mathbf{1}$. Initially *error* is false. When an overflow occurs, *error* will become true in the next state. Note that for all three approaches Theorems 1, 2, and 3 still hold.

In all versions of SMV out-of-bounds errors are checked statically which means that even unreachable out-of-bound transitions are reported as out-of-bounds errors. By using alternative 3 presented above one can check if an out-of-bounds error is reachable and report an out-of-bounds error only when an out-of-bounds error occurs on some execution path. One can also implement the

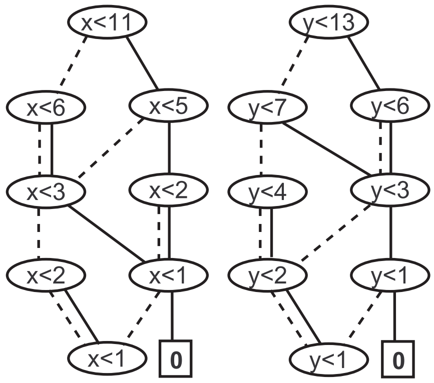

Fig. 8. Bounds information for $0 \le x < 11$ and $0 \le y < 13$

static out-of-bounds error check used in SMV by reporting a potential out-of-bounds error if a node with the *index* of the boolean variable *error′* appears in the transition relation BDD.

7 Handling multiple bounds on variables

So far we have studied the construction of BDDs for linear arithmetic constraints on v integer variables $x_i, 1 \le i \le v$ such that $0 \le x_i < 2^b$, i.e. all variables were non-negative and bounded by the same power of two. Now consider the case where each variable x_i has its own general bounds $l_i \le x_i < h_i$, where l_i and h_i are (possibly negative) integer constants. As a first step we can eliminate the lower bounds by replacing every variable x_i in every constraint by the variable $X_i = x_i - l_i$. Now any constraint of the form $\sum_{i=1}^{v} a_i \cdot x_i \# a_0$, where $\# \in \{=, \neq, >, \geq, \leq, <\}$, becomes $\sum_{i=1}^{v} a_i \cdot X_i \# a_0 - \sum_{i=0}^{v} a_i \cdot l_i$ and $0 \le X_i < h_i - l_i = d_i$. Now the lower bound of all variables is again 0 as it was initially but the upper bounds are different and not necessarily powers of two.

Here we will show how to construct BDDs for equations of the form $\sum_{i=1}^{v} a_i \cdot x_i = a_0$, where $0 \le x_i < d_i$ for $1 \le i \le v$. The construction of BDDs for inequations is similar. Since there are extra constraints $0 \le x_i < d_i$ that have to be satisfied in order for the BDD to evaluate to **1**, extra information has to be "stored" in the nodes: the valid range for the part of every variable that has not yet been processed. Since the lower bound for every variable is 0, only the upper bound needs to be stored. At the root node, the upper bound for each variable x_i is d_i. After the least significant bit $x_{i,0}$ of variable x_i has been processed, the upper bound for the rest of x_i (i.e. the value of x_i with the least significant bit removed) becomes $\lceil (d_i - x_{i,0})/2 \rceil$. In general, if the upper bound u for x_i at a node n in level i of layer j is d, then at $n.low$ $u = \lceil d/2 \rceil$ and at $n.high$ $u = \lceil (d-1)/2 \rceil$. As an example, consider the equation $2x - 3y = 1$, where $0 \le x < 11$ and $0 \le y < 13$. Figure 8 shows the bounds information for x and y as described earlier. Figure 9 shows the complete BDD for the equation.

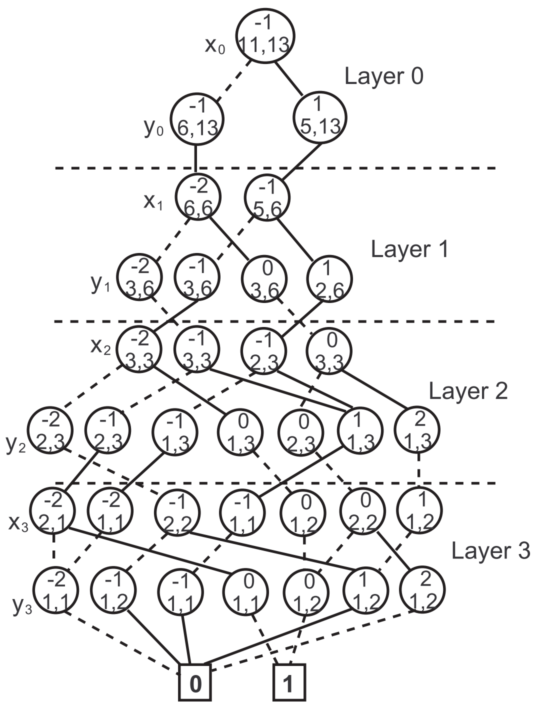

Fig. 9. BDD for $2x - 3y = 1$ when $0 \leq x < 11$ and $0 \leq y < 13$

We can prove that at any level there are at most two different upper bounds, which differ by one, for each variable. Initially, there is only one bound for each variable. If at some level the two different upper bounds for a variable are d and $d+1$, then in the next layer those bounds will become $\lceil (d-1)/2 \rceil$ and $\lceil (d+1)/2 \rceil$. In general, there can be at most 2^v different combinations of bounds for all v variables at any level. The maximum number of layers is $\log_2(\max(d_i))$. Consequently, the size of the BDD representing the equation is $O(v \cdot \log_2(\max(d_i)) \cdot 2^v \cdot \sum_{i=1}^{v} |a_i|)$, which is exponential in the number of variables v. Remember that when the upper bounds are powers of two 2^b the size of the BDD is only $O(v \cdot b \cdot \sum_{i=1}^{v} |a_i|)$. Interestingly, this indicates that when modeling a system and the choice of bounds for the integer variables is independent of the input specification, it is better to choose bounds that are powers of two.

8 Experimental results

We integrated our construction algorithms to Composite Symbolic Library and Action Language Verifier [6,13]. Action Language Verifier is an infinite state CTL model checker and it uses Composite Symbolic Library as its symbolic manipulator. We created a new version of the Action Language Verifier (ALV)

Table 1. Bakery

bits	CMU SMV	Cadence SMV	NuSMV	ALV
4	0.04	0.17	0.07	0.17
5	0.23	0.27	0.26	0.17
6	1.27	0.5	1.71	0.17
7	9.37	1.39	20.52	0.18
8	78.87	6.12	142.82	0.18
9	673.11	21.67	1186.45	0.18
10	↑	84.1	↑	0.19
11	↑	329.93	↑	0.19
12	↑	1503.83	↑	0.19
100	↑	↑	↑	0.31

Table 2. Barber

bits	CMU SMV	Cadence SMV	NuSMV	ALV
4	0.15	0.36	0.3	0.23
5	0.46	0.86	1.09	0.25
6	2.03	2.97	13.47	0.29
7	14.14	10.42	1185.92	0.3
8	274.89	38.29	↑	0.35
9	↑	157.58	↑	0.39
10	↑	721.25	↑	0.42
11	↑	↑	↑	0.44
12	↑	↑	↑	0.48
100	↑	↑	↑	5.12

by using BDDs as symbolic representations for bounded integers and integrating our BDD construction algorithms for linear arithmetic constraints.

We experimented with two specification examples, `Bakery` and `Barber`, available at: http://www.cs.ucsb.edu/˜bultan/composite/. `Bakery` is a mutual exclusion protocol for 2 processes. `Barber` is a monitor specification for the Sleeping Barber problem from [2]. We also verified three specification examples, `abp`, `p-queue` and `prod-cons`, included in the NuSMV distribution. Example `abp` is an alternating bit protocol, and `p-queue` and `prod-cons` are two different implementations of a buffer where data is inserted, sorted and consumed. We were able to verify safety and liveness properties for these examples. We run these experiments using three different implementations of the SMV, namely CMU SMV (version 2.5.4.3), Cadence SMV (version 08-20-01p2) and NuSMV (version 2). We obtained the experimental results on a SUN ULTRA 10 work station with 768 Mbytes of memory, running SunOs 5.7.

We measured the time required to verify each of the examples for different sizes of the integer variables from 4 bits to 100 bits. The results are shown in Tables 1-5. Entries ↑ signify that the according experiment did not finish in 4000 seconds. It is clear that for all current implementations of SMV, the recorded times are exponential in the size of the integer variables, while for ALV which uses the construction algorithms presented earlier, the recorded times are linear in the size of the integer variables. Note that ALV is not a BDD based model checker, hence SMV may be better optimized for BDD based verification. However, our point is that the advantages of our construction algorithms can be exploited by integrating them to any BDD based model checker.

Finally, Figure 10 illustrates the effect of arbitrary bounds on variables as described in Section 7. We used our construction algorithm to build a BDD for the equation $x_1 + x_2 - x_3 - x_4 + x_5 - x_6 = 7$, where $0 \leq x_1, x_2, x_3, x_4, x_5, x_6 < 2^8$ and recorded the size of the resulting reduced BDD. Then we gradually changed the upper bound of each of the variables to some arbitrary unique value less than 2^8 and recorded the size of the resulting reduced BDD each time. The results

Table 3. Alternating Bit Protocol. The property checked is independent of the integer data field. Since Cadence SMV can detect and verify data independent properties efficiently, we also verified a data dependent property. The verification time for the data dependent property stays almost the same for the Action Language Verifier.

bits	CMU SMV	Cadence SMV data independent	Cadence SMV data dependent	NuSMV	ALV data independent	ALV data dependent
4	0.12	0.21	3.91	2.69	0.23	0.24
5	0.26	0.2	7.56	2.71	0.23	0.25
6	1.26	0.23	24.11	2.63	0.24	0.24
7	30.24	0.18	84.33	3.05	0.23	0.26
8	147.96	0.19	343.47	3.61	0.23	0.24
9	693.67	0.2	↑	6.6	0.25	0.26
10	3755.46	0.24	↑	24.12	0.23	0.27
11	↑	0.27	↑	87.62	0.27	0.27
12	↑	0.36	↑	342.89	0.24	0.27
100	↑	↑	↑	↑	0.69	0.72

Table 4. Queue

bits	CMU SMV	Cadence SMV	NuSMV	ALV
4	0.3	0.57	0.33	3.21
5	1.75	1.23	1.21	5.63
6	24.47	5.37	12.07	8.14
7	2159.69	38.8	122.3	10.77
8	↑	318.39	1125.34	13.06
100	↑	↑	↑	440.87

Table 5. Producer - Consumer

bits	CMU SMV	Cadence SMV	NuSMV	ALV
2	5.49	4.61	23.27	210.44
3	216.94	73.98	3264.97	698.93
4	↑	1430.54	↑	2600
5	↑	↑	↑	6062.34

shown in Figure 10 demonstrate the exponential growth of the size of the BDD described in Section 7.

9 Conclusions

In this paper we have shown experimentally that current implementations of BDD based symbolic model checkers are inefficient in representing linear arithmetic constraints on bounded integer variables. We solved this problem by giving efficient BDD construction algorithms, proving their complexity and experimentally demonstrating their efficiency. These algorithms can be used to improve the performance of existing BDD based symbolic model checkers. Finally, we have shown that powers of 2 are a good choice for variable bounds, and choosing arbitrary bounds for integer variables can cause exponential blow-up in the BDD size.

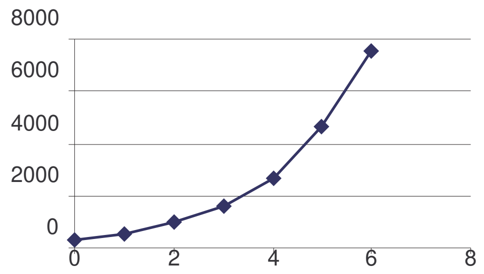

Fig. 10. Size of the BDD for $x_1 + x_2 - x_3 - x_4 + x_5 - x_6 = 7$ versus the number of variables with upper bound different than 2^8

References

1. Cadence SMV. `http://www-cad.eecs.berkeley.edu/~kenmcmil/smv`.
2. G. R. Andrews. *Concurrent Programming: Principles and Practice*. The Benjamin/Cummings Publishing Company, Redwood City, California, 1991.
3. R. Bryant. Graph-based algorithms for boolean function manipulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, 1990.
4. R.E Bryant and Y.A. Chen. Verification of arithmetic functions with binary moment diagrams. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, June 1995.
5. T. Bultan, R. Gerber, and C. League. Composite model checking: Verification with type-specific symbolic representations. *ACM Transactions on Software Engineering and Methodology*, 9(1):3–50, January 2000.
6. T. Bultan and T. Yavuz-Kahveci. Action language verifier. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, 2001.
7. W. Chan, R. J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. D. Reese. Model checking large software specifications. *IEEE Transactions on Software Engineering*, 24(7):498–520, July 1998.
8. A. Cimatti, E.M. Clarke, E.Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proceedings of the International Conference on Computer-Aided Verification*, 2002.
9. E.M. Clarke, M. Fujita, and X. Zhao. Hybrid decision diagrams - overcoming the limitations of mtbdds and bmds. In *In International Conference of Computer-Aided Design*, pages 159–163, 2000.
10. M. Garey and D.Jonson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, 1979.
11. K. L. McMillan. *Symbolic model checking*. Kluwer Academic Publishers, Massachusetts, 1993.
12. J. Yang, A. K. Mok, and F. Wang. Symbolic model checking for event-driven real-time systems. *ACM Transactions on Programming Languages and Systems*, 19(2):386–412, March 1997.
13. 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.