The Constraint Database Approach to Software Verification*

Peter Revesz

Max Planck Institut für Informatik
University of Nebraska-Lincoln
revesz@cse.unl.edu

Abstract. Based on constraint database techniques, we present a new approach to software verification. This new approach has some similarity to abstract interpretation that uses various widening operators; therefore, we call the new approach l-u widening. We show that our l-u widening leads to a more precise over-approximation of the invariants in a program than comparable previously proposed widening operators based on difference-bound matrices, although l-u widening can be computed as efficiently as the other widening operators. We show that constraint database techniques can compute non-convex program invariants too. Finally, we give a compact representation of addition-bound matrices, which generalize difference-bound matrices.

1 Introduction

Software verification is a basic concern of computer science, hence many different approaches were proposed for it, including data flow analysis, abstract interpretation [**?**,**?**,**?**], model checking [**?**,**?**,**?**,**?**], predicate abstraction [**?**], and mathematical induction. Today there are many examples of successful applications of these approaches to the verification of digital circuits and programs.

Software verification would be easy if we could compute the precise semantics of programs. For a procedural program, the semantics means that we find for each line of the program an invariant, which is the set of possible values of the variables that may be used at that line. While a precise computation is not possible in general, an over-approximation or under-approximation is possible. Abstract interpretation relies on a kind of over-approximation. More recently, constraint database researchers proposed for *constraint query languages* [**?**,**?**,**?**,**?**,**?**], which simplify *constraint logic programs* [**?**,**?**,**?**,**?**], alternative methods to over-approximate or under-approximate the semantics [**?**,**?**]. Via well-known translations among the various programming languages, the approximation results in constraint databases imply approximation results for the semantics of the more traditional procedural programs. The idea of translating from procedural programs to constraint query languages or constraint logic programs occurs in Delzanno and Podelski [**?**], Fribourg and Richardson [**?**], and Fribourg and Olson [**?**]. However, these papers did not use the latest approximation results. For example, [**?**] relied on the result that the least fixpoint semantics of Datalog (Prolog without function symbols and negation) with integer gap-order constraint programs can be precisely evaluated [**?**].[1]

In this paper we present a general approach of applying the constraint database approximation to software verification, extending earlier work in [**?**]. The constraint database approximations are

* This research was supported in part by a Humboldt Research Fellowship from the Alexander von Humboldt Foundation.

[1] A gap-order is a constraint of the form $x - y \geq c$ or $\pm x \geq c$ where x and y are integer or rational variables and c is a non-negative integer constant.

different from *abstract interpretation* methods, which seem closest to them among the well-known software verification approaches. To further clarify their relationships, we introduce a new method between the constraint database approximations in [**?**,**?**] and abstract interpretation. We call this new method *l-u-widening*. We show that *l-u widening* is more precise than other widening operators proposed for abstract interpretation. On the other hand, program semantics approximations based on *l-u widening* can be more efficiently computed than program semantics approximations based on the constraint database techniques in [**?**,**?**] can be computed.

The rest of this paper is organized as follows. Section **??** gives a brief review of constraints, abstract interpretation, and difference bound matrices. It also describes addition-bound matrices which are similar to difference-bound matrices. Section **??** presents our new *l-u widening* operator and its use in approximating the semantics of programs. Section **??** reviews the earlier constraint database approximation methods and applies them to some sample programs. Section **??** presents an outline of the constraint database approach to software verification. Section **??** describes a novel compact representation of addition-bound matrices. This representation can be efficient for computer implementations. Finally, Section **??** discusses some related and future work.

2 Basic Concepts

2.1 Constraints

$$
\begin{array}{l}
\forall a, b, S_1, S'_1, \ldots S_n, S'_n \\
\quad (\ \mid S_1 \mid + \ldots + \mid S_n \mid = n \wedge \\
\qquad (\forall i, j, k\ \ ((i \neq j \wedge i \neq k \wedge j \neq k \wedge 1 \leq i, j, k \leq n) \rightarrow \\
\qquad\quad ((a \in S_i\ \wedge\ b \in S_j)\ \rightarrow (S'_i = \emptyset\ \wedge\ S'_j = S_i \cup S_j\ \wedge\ S'_k = S_k)))) \\
\qquad \rightarrow\ \ \mid S'_1 \mid + \ldots + \mid S'_n \mid = n)
\end{array}
$$

We use the following basic or *atomic constraints*:

Lower Bound :	$x \geq b$
Upper Bound :	$-x \geq b$
Difference :	$x - y \geq b$
Addition :	$\pm x \pm y \geq b$
Linear :	$c_1 x_1 + \ldots + c_n x_n \geq b$

where x, y and the x_is are integer or rational variables and b, called the *bound*, and the c_is are integer constants.

Note: For uniformity, we prefer to always use constraints that end with "$\geq b$." We make some exceptions when other forms are clearer. For example, we use equalities of the form $x = b$ as a shorthand for $(x \geq b) \wedge (-x \geq -b)$. Some authors use the terms *potential constraint* and *sum constraint*. A *potential constraint* of the form $x - y \leq b$ translates to the difference constraint $y - x \geq -b$, and a *sum constraint* of the form $\pm x \pm y \leq b$ translates to the addition constraint $\pm x \pm y \geq -b$ with changed signs for x and y. Therefore, any result on potential constraints and sum constraints can be trivially translated to results on difference or addition constraints and vice versa.

2.2 Abstract Interpretation

Abstract interpretation finds *invariants* associated with specific program locations, such that each invariant is an over-approximation of the set of possible values of the program variables at that location, and that the invariants cannot be extended further by additional abstract execution of the program. Each invariant can be compactly described as some constraint on the program variables, for example, conjunctions of linear equations and inequalities, if the program variables are all rational numbers.

Abstract interpretation methods typically use a *widening operator*. Common widening operators use the domains of intervals [**?**] or polyhedra [**?**,**?**]. During an abstract execution of the program, the widening operator repeatedly updates a constraint M that describes the current value of the invariant associated with a program location with a new constraint N that describes an additional set of possible values of the program variables at that location. This happens when due to some program loop we reenter the same location again.

To keep things computationally feasible, the widening operator cannot just take $M \cup N$ as the new value of the invariant. Instead, it calculates a *convex* region, that is, a conjunction of linear inequality constraints that includes both M and N. In addition, when we use widening operators, we need to avoid an infinite number of repeated applications of the widening operators. The following clever idea guarantees that: *Preserve those constraints of M that are implied by N.* This looks attractive, because if M contains k linear inequalities, then at most k widening operators can be performed on M.

2.3 Addition-Bound Matrices

Miné [**?**] represents a conjunction of lower bound, upper bound, and sum constraints over variables $V = \{x_1, \ldots, x_n\}$ by a conjunction of potential constraints over variables $V^+ = \{x_1^+, x_1^-, \ldots, x_n^+, x_n^-\}$, that is, every variable has a positive form x_i^+ equivalent to x_i and a negative form x_i^- equivalent to $-x_i$.

Rephrasing Miné's idea, a conjunction of lower bound, upper bound and addition constraints C over variables $V = \{x_1, \ldots, x_n\}$ can be represented by a conjunction of difference constraints over variables $V^+ = \{x_1^+, x_1^-, \ldots, x_n^+, x_n^-\}$, as follows:

$$\begin{array}{rcl} x \geq b & \equiv & x^+ - x^- \geq 2b \\ -x \geq b & \equiv & x^- - x^+ \geq 2b \\ x + y \geq b & \equiv & x^+ - y^- \geq b \\ x - y \geq b & \equiv & x^+ - y^+ \geq b \\ -x + y \geq b & \equiv & x^- - y^- \geq b \\ -x - y \geq b & \equiv & x^- - y^+ \geq b \end{array}$$

Now a conjunction of difference constraints can be simplified as follows. If the conjunction contains two difference constraints of the form $x - y \geq b$ and $x - y \geq c$ where $b > c$, then we can delete the second constraint, because it is already implied by the first constraint. By this simplification, there is at most one constraint with the left hand hide $x - y$, for any pair of variables x and y. We apply this simplification to the conjunction of difference constraints that result after our translation.

The conjunction of difference constraints C over variables $\{x_1, \ldots, x_n\}$ can be represented by an $n \times n$ *Addition-Bound Matrix* M, which is defined as follows:

$$M[i,j] = \left\{ \begin{array}{ll} b & \text{if } (x_i - x_j \geq b) \in C \\ -\infty & \text{otherwise} \end{array} \right\}$$

Note: Rather confusingly, it is common to call *Difference-Bound Matrices (DBMs)* those matrices that represent conjunctions of potential constraints C and are actually defined as having entry b if $x_i - x_j \leq b$ is in C and $+\infty$ otherwise. We use the term *Addition-Bound Matrix (ABM)* because we ultimately represent by ABMs conjunctions of addition, lower bound, and upper bound constraints over V, although not directly as we first translate these constraints to conjunctions of difference constraints over V^+.

Example 1. Consider the following conjunction of lower bound, upper bound, and addition constraints over the variables x and y:

$$-x \geq -25, \quad y \geq 3, \quad x - y \geq 4, \quad x + y \geq 10, \quad -x - y \geq -40$$

These can be translated into the following difference constraints over the variables x^+, x^-, y^+, y^-:

$$x^- - x^+ \geq -50, \quad y^+ - y^- \geq 6, \quad x^+ - y^+ \geq 4, \quad x^+ - y^- \geq 10, \quad x^- - y^+ \geq -40$$

This set of difference constraints can be represented by the following ABM:

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	-50	$-\infty$	-40	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	6
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

This simple representation of ABMs will suffice to describe the main theorems of the paper. Later in Section **??**, we outline a more compact ABM representation that may lead to a more efficient computer implementation.

2.4 Operations on ABMs

Next we define some basic operators on ABMs.

Definition 1. Let M and N be two ABMs. Then the min of M and N, written as $M \vee N$, is defined as follows.

$$[M \vee N]\,[i,j] = \min(M[i,j], N[i,j])$$

Alternatively, we can write the above as:

$$[M \vee N]\,[i,j] = \left\{ \begin{array}{ll} M[i,j] & \text{if } M[i,j] \leq N[i,j] \\ N[i,j] & \text{if } N[i,j] \leq M[i,j] \end{array} \right\}$$

Miné's widening operator on DBMs [**?**] can be rephrased on ABMs as follows.

Definition 2. Let M and N be two ABMs. Then the *widening* of M by N, written as $M \triangledown N$, is defined as follows.

$$[M \triangledown N]\,[i,j] = \left\{ \begin{array}{ll} M[i,j] & \text{if } M[i,j] \leq N[i,j] \\ -\infty & \text{if } N[i,j] < M[i,j] \end{array} \right\}$$

Example 2. Let M be as in Example **??**, and let N be the following ABM:

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	15	10
x^-	-60	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	7	$-\infty$	2
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

In this case $M \vee N$ is:

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	-60	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	2
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

while $M \triangledown N$ is:

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	$-\infty$
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

3 The l-u-Widening Operator

We say matrix M has domain D if all entries of M are in D. If all entries of M are $\geq l$ and $\leq u$ or $-\infty$, where l and u are some integer constants, then the domain of M is $\{-\infty\} \cup \{l, l+1, \ldots, u-1, u\}$. (The domain of M should not be confused with the domain of the variables which are integer or rational numbers.)

We now introduce the *l-u-widening* operator.

Definition 3. Let $l < 0$ and $u > 0$ be two integer numbers. Let M and N be two ABMs such that the domain of M is $\{-\infty\} \cup \{l, l+1, \ldots, u-1, u\}$. Then the *l-u-widening* of M by N, written as $M \Diamond_{l,u} N$, is defined as follows.

$$[M \Diamond_{l,u} N]\,[i,j] = \left\{ \begin{array}{ll} M[i,j] & \text{if } \; M[i,j] \leq N[i,j] \\ N[i,j] & \text{if } \; l \leq N[i,j] < M[i,j] \\ -\infty & \text{if } \; N[i,j] < l \leq M[i,j] \end{array} \right\}$$

Example 3. Let us continue Example **??** and find $M \Diamond_{-50,50} N$, the *l-u-widening* of M and N with $l = -50$ and $u = 50$.

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	2
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

3.1 Properties of l-u-Widening

In this section we compare the precision of the widening and l-u-widening operators. Let $\mathcal{S}$ be the solution space of an ABM or union of ABMs. We have the following.

Theorem 1. For any $l < 0$ and $u > 0$, the following holds:

$$\mathcal{S}(M \cup N) \subseteq \mathcal{S}(M \vee N) \subseteq \mathcal{S}(M \Diamond_{l,u} N) \subseteq \mathcal{S}(M \triangledown N).$$

Proof. By Definition **??**, each entry of $M \vee N$ is smaller than the corresponding entry in either M or N. Hence $\mathcal{S}(M) \subseteq \mathcal{S}(M \vee N)$ and $\mathcal{S}(N) \subseteq \mathcal{S}(M \vee N)$ hold. Therefore, $\mathcal{S}(M \cup N) \subseteq \mathcal{S}(M \vee N)$ also must hold.

By Definition **??**, the *minimum*, *widening* and *l-u-widening* operators behave the same when $M[i,j] \leq N[i,j]$. When $N[i,j] < M[i,j]$ then there are two cases. In the first case, when $l \leq N[i,j]$, then the ł-u-widening operator behaves like the *minimum* operator and returns $N[i,j]$, and if $N[i,j] < l$, then it behaves like the *widening* operator and returns $-\infty$. Therefore, $\mathcal{S}(M \vee N) \subseteq \mathcal{S}(M \Diamond_{l,u} N) \subseteq \mathcal{S}(M \triangledown N)$ must hold. □

Definition 4. Given a program P, and values $l < 0$ and $u > 0$, the result of evaluating its least fixed point using l-u-widening is written as $W^{l,u}$.

The following is the main l-u-approximation theorem.

Theorem 2. Let $l < 0$ and $u > 0$ be integer constants. For any program P with m lines and n variables the following holds.

$$\mathit{lfp}(P) \ \subseteq \ W^{l,u}$$

where $\mathit{lfp}(P)$ is the least fixed point of P. Further, $W^{l,u}$ can be computed using $O(|u-l|mn^2)$ time.

Proof. We start evaluating P. For each new line L_i of P, when we find the first ABM for it, we change all entries greater than u to u and call the resulting ABM M_i. Then whenever a new ABM N is found for line L_i, we update M_i to be the result of $M_i \Diamond_{l,u} N$. This ensures that the domain of each M_i is $\{-\infty\} \cup \{l, l+1, \ldots, u-1, u\}$ throughout the approximate evaluation. In each iteration at least one entry in at least one of the M_is must decrease. Moreover, each entry can decrease at most $|u-l|$ times and each M_i has n^2 entries. Since there are m number of M_is, the total number of iterations is at most $|u-l|mn^2$. It is also clear that the approximate evaluation is always computing an upper approximation of the actual least fixed point. □

The computational complexity of the l-u-widening operator is similar to that of Miné's widening operator, which needs $O(mn^2)$ iterations. If the values of u and l are fixed constants, then the use of the two widening operators will have the same complexity. However, there are reasons to vary the values of l and u, because we can also get tighter approximations using increasingly smaller values of l or larger values of u. That is, we can show the following.

Theorem 3. For each program P and constants l_1, l_2, u_1, and u_2 such that $l_1 \leq l_2 < 0 < u_2 \leq u_1$, the following condition holds:

$$W^{l_1,u_1} \ \subseteq \ W^{l_2,u_2}$$

3.2 A Simple Program with Goto Statements

Consider the following simple program fragment.

```
1 a = 0
2 a = a + 1
3 if a > 2 then  goto 6
4 if a = 2 then  goto 7
5 goto 2
6
7
```

Let us see how this widening operator works on this program. Let $L_{i,j}$ be the invariant at the beginning of line i at the jth entry of that line. Initially all $L_{i,0}$ are empty. $L_{2,1} = \{a = 0\}$ (which, like all equalities, is just a shorthand for a conjunction of two inequalities, namely in this case $\{0 \leq a \leq 0\}$), and $L_{3,1} = \{a = 1\}$. Lines 3 and 4 have false if conditions and do not change the value of a, hence $L_{4,1} = L_{5,1} = \{a = 1\}$. The execution of line 5 takes us back to the beginning of line 2 with no change in a. This is the second entry to line 2, hence $L_{2,2} = L_{2,1} \triangledown L_{5,1} = \{a = 0\} \triangledown \{a = 1\} = \{a \geq 0\}$. When $a = a + 1$ is executed, this yields $a \geq 1$. We enter line 3 for the second time. By widening we get $L_{3,2} = L_{3,1} \triangledown \{a \geq 1\} = \{a = 1\} \triangledown \{a \geq 1\} = \{a \geq 1\}$. We enter the if statement and find that $L_{3,2} \wedge (a > 2) = (a \geq 1) \wedge (a > 2) = a > 2$. That is, our invariant (or rather our current best estimate of the possible values of the program variable a at the beginning of line 3) and the condition of the if statement overlap on $a > 2$, which clearly is a nonempty set. Hence we enter line 6 with $L_{6,1} = \{a > 2\}$. This example can be summarized in the table below.

Invariants Obtained by Widening

Line	1st Entry	2nd Entry
2	$0 \leq a \leq 0$	$0 \leq a$
3	$1 \leq a \leq 1$	$1 \leq a$ if condition $a > 2$ true goto 6
4	$1 \leq a \leq 1$	
5	$1 \leq a \leq 1$	

However, the above program analysis is wrong. Actually, the program can never enter line 6. When we first get to line 5, $a = 1$. Hence when we get back to line 2 and execute $a = a + 1$, then $a = 2$. Therefore the if condition of line 3 will fail, and the program goes on to line 4. The if condition of line 4 will be true, hence we go to line 7 and never enter line 6.

Clearly, the invariant analysis is not precise enough. The inductive generalization that the widening operator applies (for example, in the above program from $a = 0$ and $a = 1$ to $a \geq 0$) is often very useful and powerful, but it has to be applied more judiciously. At the present, there are only some limited techniques in the abstract interpretation area to get around the above problem. For example, we may establish a priori a set of constraints K and widen M up-to K only, but finding a suitable K is easier said than done. For example, if K contains $\{a \leq 3\}$, then we may widen $a = 0$ and $a = 1$ to $0 \leq a \leq 3$, but then the program analysis would be still incorrect.

Now let us see how the *l-u widening* works on the same program with $l = -5$ and $u = 5$. The crucial difference is that on the second entry to line 2, we obtain $L_{2,2} = L_{2,1} \Diamond_{-5,5} L_{5,1} = \{a = 0\} \Diamond_{-5,5} \{a = 1\} = \{0 \leq a \leq 1\}$. When $a = a + 1$ is executed, this yields $1 \leq a \leq 2$. We enter line 3 for the second time and get $L_{3,2} = L_{3,1} \Diamond_{-5,5} \{1 \leq a \leq 2\} = \{1 \leq a \leq 2\}$. We enter the if statement but find that $L_{3,2} \wedge (a > 2)$ is unsatisfiable. Therefore. we continue to line 4 and find that $L_{4,2} = L_{4,1} \Diamond_{-5,5} \{1 \leq a \leq 2\} = \{1 \leq a \leq 2\}$. W enter the if statement and find that $L_{4,2} \wedge a = 2$ is satisfiable. Hence we go to line 7. The invariants found by the *l-u widening* are summarized in the table below.

Invariants Obtained by l-u Widening

Line	**1st Entry**	**2nd Entry**
2	$0 \leq a \leq 0$	$0 \leq a \leq 1$
3	$1 \leq a \leq 1$	$1 \leq a \leq 2$ if condition $a > 2$ false
4	$1 \leq a \leq 1$	$1 \leq a \leq 2$ if condition $a = 2$ true goto 7
5	$1 \leq a \leq 1$	

3.3 The Subway Train Example

Let us consider the following subway train speed regulation system described by Halbwachs [**?**]. Each train detects beacons that are placed along the track and receives a "second" signal from a central clock.

Let b and s be counter variables for the number of beacons and second signals received. Further, let d be a counter variable that describes how long the train is applying its brake. The goal of the speed regulation system is to keep $\mid b - s \mid$ small while the train is running.

The speed of the train is adjusted as follows. When $s + 10 \leq b$, then the train notices it is early and applies the brake as long as $b > s$. Continuously braking causes the train to stop before encountering 10 beacons.

When $b + 10 \leq s$ the train is late and will be considered late as long as $b < s$. As long as any train is late, the central clock will not emit the second signal.

The following program implements the subway train regulation using parallel case statements. In a parallel case statement one of the cases is selected randomly. If the condition of the selected case statement is false, then another is selected and executed. This repeats until one of the cases succeeds.

Train(b,s,d)

```
1 ONTIME
  begin parallel
2       if b − s > −9 then s = s + 1 goto ONTIME
3       if b − s = −9 then s = s + 1 goto LATE
4       if b − s < 9  then b = b + 1 goto ONTIME
5       if b − s = 9  then b = b + 1 goto BRAKE
```

end parallel
6 LATE
begin parallel
7 **if** $b - s < -1$ **then** $b = b + 1$ **goto** LATE
8 **if** $b - s = -1$ **then** $b = b + 1$ **goto** ONTIME
end parallel
9 STOPPED
begin parallel
10 **if** $b - s > 1$ **then** $s = s + 1$ **goto** STOPPED
11 **if** $b - s = 1$ **then** $s = s + 1$ **goto** ONTIME
end parallel
12 BRAKE
begin parallel
13 **if** $b - s > 1$ **then** $s = s + 1$ **goto** BRAKE
14 **if** $b - s = 1$ **then** $s = s + 1,\ d = 0$ **goto** ONTIME
15 **if** $d < 9$ **then** $b = b + 1,\ d = d + 1$ **goto** BRAKE
16 **if** $d \leq 9$ **then** $b = b + 1,\ d = 0$ **goto** STOPPED
end parallel

Suppose we know that the subroutine $Train$ can be called with any values where $b = s$ and $d = 0$. We need to find all the possible values of the variables b, s and d in all lines of the program.

Note that variable d is changed only in the parallel case statement after BRAKE. When we exit the BRAKE region and go to either ONTIME or STOPPED, then d is reset to 0. Hence d always remains 0 outside of the BRAKE region. This simplifies the analysis for the other three cases. With only variables b and s, each conjunction of difference constraints can be represented in the form:

$$c_1 \leq b \leq c_2, \quad c_3 \leq s \leq c_4, \quad c_5 \leq b - s \leq c_6$$

where $c_1,\ c_2,\ c_3,\ c_4,\ c_5,\ c_6$ are constants that may be omitted.

$L_{1,1} = \{0 \leq b - s \leq 0\}$
line 2 causes return to ONTIME with $\{-1 \leq b - s \leq -1\}$
line 3 fails
line 4 causes return to ONTIME with $\{1 \leq b - s \leq 1\}$
line 5 fails
$L_{1,2} = L_{1,1} \triangledown (\{-1 \leq b - s \leq -1\} \sqcup \{1 \leq b - s \leq 1\}) = \{-1 \leq b - s \leq 1\}$

$\vdots$
$L_{1,9} = \{-9 \leq b - s \leq 9\}$
line 2 causes return to ONTIME with $\{-9 \leq b - s \leq 8\}$
line 3 causes entry to LATE with $\{-10 \leq b - s \leq -10\}$
line 4 causes return to ONTIME with $\{-8 \leq b - s \leq 9\}$
line 5 causes entry to BRAKE with $\{10 \leq b - s \leq 10\}$
$L_{1,10} = L_{1,9}$

$L_{6,1} = \{-10 \leq b - s \leq -10\}$
line 7 causes return to LATE with $\{-9 \leq b - s \leq -9\}$

line 8 fails
$L_{6,2} = L_{6,1} \triangledown \{-9 \leq b - s \leq -9\} = \{-10 \leq b - s \leq -9\}$
$\vdots$
$L_{6,10} = \{-10 \leq b - s \leq -1\}$
line 7 causes return to LATE with $\{-10 \leq b - s \leq -2\}$
line 8 causes return to ONTIME with $\{-1 \leq b - s \leq -1\}$
$L_{6,11} = L_{6,10}$
$L_{1,11} = L_{1,10}$

$L_{12,1} = \{10 \leq b - s \leq 10,\ d = 0\}$
line 13 causes return to BRAKE with $\{9 \leq b - s \leq 9,\ d = 0\}$
line 14 fails
line 15 causes return to BRAKE with $\{11 \leq b - s \leq 11,\ d = 1\}$
line 16 causes entry to STOPPED with $\{11 \leq b - s \leq 11,\ d = 0\}$
$L_{12,2} = L_{12,1} \triangledown (\{9 \leq b - s \leq 9,\ d = 0\} \sqcup \{11 \leq b - s \leq 11,\ d = 1\} = \{9 \leq b - s \leq 11,\ 0 \leq d \leq 1\}$

$\vdots$line 16 causes entry to STOPPED with $\{2 \leq b - s \leq 20, d = 0\}$
$L_{12,10} = \{1 \leq b - s \leq 19,\ 0 \leq d \leq 9\}$

$L_{9,1} = \{2 \leq b - s \leq 20, d = 0\}$
line 10 causes return to STOPPED with $\{1 \leq b - s \leq 19, d = 0\}$
line 11 fails
$L_{9,2} = L_{9,1} \triangledown \{1 \leq b - s \leq 19, d = 0\} = \{1 \leq b - s \leq 20, d = 0\}$
line 10 causes return to STOPPED with $\{1 \leq b - s \leq 19, d = 0\}$
line 11 causes return to ONTIME with $\{0 \leq b - s \leq 0, d = 0\}$
$L_{9,3} = L_{9,2}$
$L_{1,11} = L_{1,10}$

The table below shows the result of the invariants that can be found using $l = -20$ and $u = 20$.

Invariants Obtained by l-u Widening

Brake	Late	Ontime	Stopped
$1 \leq b - s \leq 19$ $0 \leq d \leq 9$	$-10 \leq b - s \leq -1$ $0 \leq d \leq 0$	$-9 \leq b - s \leq 9$ $0 \leq d \leq 0$	$1 \leq b - s \leq 20$ $0 \leq d \leq 0$

It is possible to prove that these values match the actual semantics of the program.

4 Non-Convex Invariants

In the constraint database area, researchers have founds methods for finding over-approximations and under-approximations of the least fixpoint semantics of Datalog programs. The over-approximation yields for each relation a disjunction of conjunctions of atomic constraints. In this sense the approximation is different from widening operators that always yield a conjunction of atomic constraints.

Definition 5. Given any conjunction $\mathcal{C}$ of addition constraints and integer $l < 0$, let $\mathcal{C}'$ be the result of deleting from $\mathcal{C}$ any constraint where the bound is less than l. Further, let $\mathcal{C}''$ be the result of replacing in $\mathcal{C}$ any bound with less than l with l.

It is easy to see that $\mathcal{C}'$ is an over-approximation and $\mathcal{C}''$ is an under-approximation of $\mathcal{C}$. Further, this leads to the following evaluation idea.

Definition 6. Given a program P and value $l < 0$, the result of evaluating its least fixed point by always rewriting after each rule application any conjunction of constraints $\mathcal{C}$ into a $\mathcal{C}'$ (or $\mathcal{C}''$) as in Definition **??** is written as $P^{l,u}$ (respectively, $P_{l,u}$).

The following is the main theorem that we can adopt.

Theorem 4. Let $l < 0$ be any integer constant. For any program P the following holds.

$$P_{l,u} \ \subseteq \ \mathit{lfp}(P) \ \subseteq \ P^{l,u}$$

Further, $P_{l,u}$ and $P^{l,u}$ can be computed in finite time.

The bottom up evaluation in Theorem **??** is slower than the *l-u widening* approach. However, it can lead to a more precise over-approximation or under-approximation than the *l-u widening approach*.

Example 4. Consider the following program.

```
1 x = 1,  y = 1
2 x = x + 1,  y = y + 2x - 1
3 goto 2
```

For the above program, it is easy to see that for each entry i of line 2, we have:

$$L_{2,i} = \{x = i,\ y = i^2\}.$$

Recall that each equality is the conjunction of a lower and an upper bound atomic constraint. That is,

$$L_{2,i} = \{x \geq i,\ -x \geq -i,\ y \geq i^2,\ -y \geq -i^2\}.$$

Hence when we evaluate the semantics of this program using $l = -10$, we obtain the following over-approximation: This formula is the union of three parts. Clearly, the first part corresponds to the actual semantics for $1 \leq i \leq 3$. The second part is an over-approximation needed because for $4 \leq i \leq 10$ we can only express the upper bounds $y \geq i^2$ but cannot express the lower bounds $-y \geq -i^2$ needed to have a precise evaluation matching the actual semantics. The third part is needed because for any $i = 11$ we can express neither the lower bound $-x \geq i$ nor the lower bound $-y \geq -i^2$. Finally, note that for any $i \geq 12$, the conjunction of the constraints $x \geq i$ and $y \geq i^2$ are more restrictive than the third part.

5 Verification

Suppose that we need to check that certain *error states* never occur during any execution of a program. The error states are expressed as a quantifier-free formula of the variables used in the program. Each satisfying assignment of values to the variables is an error that needs to be avoided. Next we outline a general constraint database approach to the verification of programs.

1. Translate the program P into a transition system T.
2. Translate T into a Datalog program that always derives conjunctions of addition constraints.
3. Find an over-approximation of the least fixed point semantics of the Datalog program.
4. Check that the over-approximation and the error states do not intersect.
5. If the answer is "yes", then return "'safe"', else goto 1 and try a smaller l.

Step (1) is well-known in the software verification area. Step (2) is explained in Chapter 5 of [**?**], to which we refer for the details. Step (3) follows Theorems **??** and **??** with more details in [**?**]. The over-approximation algorithm is implemented within the MLPQ constraint database system [**?**], which is available from the website: `cse.unl.edu/~revesz`. Step (4) requires to test the satisfiability of the over-approximation and the error states. Finally, Step (5) is just a repetition of the previous steps in case the check is inconclusive. In the MLPQ system the user can specify any negative l value.

In the above outline, the translations to transition system and to Datalog are required only to take a direct advantage of the already implemented constraint database systems such as MLPQ. Those who are familiar with abstract interpretations with widening operators may skip the translations steps and consider an invariant analysis similar to abstract interpretation with the widening operator replaced by *l-u widening* or the non-convex approximation.

Example 5. Consider again the subway train example. The Datalog with addition constraint program that is equivalent to the subway train control program is described in [**?**]. Let $\mathcal{E}$, the error states, be as follows:

$$\mathcal{E} = \{b, s \ : \ |b - s| > 20\}.$$

It can be checked that the over-approximation found for the subway train and $\mathcal{E}$ have no common solution. Hence the subway train program is safe to use.

Example 6. Consider again the program in Example **??**. This program can be translated into the following constraint Datalog program.

$$\begin{array}{ll} Line2(x, y) & :- \ x = 1, \ y = 1. \\ Line2(x', y') & :- \ Line2(x, y), \ x' = x + 1, \ y' = y + 2x' - 1. \\ Line3(x, y) & :- \ Line2(x, y), \ x' = x + 1, \ y' = y + 2x' - 1. \end{array}$$

We can calculate the over-approximation of the above Datalog program similar to Example **??**. It is interesting to see how the bottom up evaluation terminates. In the 11th application of the second rule (which corresponds to the 12th entry of line 2 in the original program), the bottom up evaluation finds that $\exists x, y \ x' = x + 1, \ y' = y + 2x' - 1, \ x \geq 11, \ y \geq 121 \ = \ x' \geq 12, y' \geq 144$. Before adding this to the set of already existing ABMs for Line2, we need to replace x' by x and y' by y. The replacement yields $x \geq 12, y \geq 144$, which is more restrictive than $x \geq 11, y \geq 121$,

the previously added ABM. Hence it is not added to the set of ABMs for $Line2$ and the evaluation terminates.[2]

Let the error states $\mathcal{E}$ be as follows:

$$\mathcal{E} = \{y \ : \ y \geq 10, \ -y \geq -15\}.$$

This is a region where y is between 10 and 15 inclusively. It can be easily checked that the over-approximation of $L_{2,1}$ in Example **??** and $\mathcal{E}$ have no common solution. Hence the program can never enter the error states.

Example 7. Consider the following program.

```
1 x = 1,  y = 1
2 x = x + 1,  y = y + 2x - 1
3 goto 2
```

For the above program, it is easy to see that for each entry i of line 2, we have:

$$L_{2,i} = \{x = i, \ y = i^2\}.$$

Recall that each equality is the conjunction of a lower and an upper bound atomic constraint. That is,

$$L_{2,i} = \{x \geq i, \ -x \geq -i, \ y \geq i^2, \ -y \geq -i^2\}.$$

Hence when we evaluate the semantics of this program using $l = -10$, we obtain the following over-approximation:

$$L_{2.i} = \{x = i, \ y = i^2 : 1 \leq i \leq 3\} \ \cup \ \{x = i, \ y \geq i^2 : 4 \leq i \leq 10\} \ \cup \ \{x \geq 11, \ y \geq 121\}$$

This formula is the union of three parts. Clearly, the first part corresponds to the actual semantics for $1 \leq i \leq 3$. The second part is an over-approximation needed because for $4 \leq i \leq 10$ we can only express the upper bounds $y \geq i^2$ but cannot express the lower bounds $-y \geq -i^2$ needed to have a precise evaluation matching the actual semantics. The third part is needed because for any $i = 11$ we can express neither the lower bound $-x \geq i$ nor the lower bound $-y \geq -i^2$. Finally, note that for any $i \geq 12$, the conjunction of the constraints $x \geq i$ and $y \geq i^2$ are more restrictive than the third part.

6 Verification

Suppose that we need to check that certain *error states* never occur during any execution of a program. The error states are expressed as a quantifier-free formula of the variables used in the program. Each satisfying assignment of values to the variables is an error that needs to be avoided. Next we outline a general constraint database approach to the verification of programs.

1. Translate the program P into a transition system T.
2. Translate T into a Datalog program that always derives conjunctions of addition constraints.

[2] This is a simplification of the bottom up evaluation, because within the ABMs constraints of the form $x \geq b$ are represented by $x^+ - x^- \geq 2b$ as described in Section **??**.

3. Find an over-approximation of the least fixed point semantics of the Datalog program.
4. Check that the over-approximation and the error states do not intersect.
5. If the answer is "yes", then return "'safe"', else goto 1 and try a smaller l.

Step (1) is well-known in the software verification area. Step (2) is explained in Chapter 5 of [**?**], to which we refer for the details. Step (3) follows Theorems **??** and **??** with more details in [**?**]. The over-approximation algorithm is implemented within the MLPQ constraint database system [**?**], which is available from the website: `cse.unl.edu/~revesz`. Step (4) requires to test the satisfiability of the over-approximation and the error states. Finally, Step (5) is just a repetition of the previous steps in case the check is inconclusive. In the MLPQ system the user can specify any negative l value.

In the above outline, the translations to transition system and to Datalog are required only to take a direct advantage of the already implemented constraint database systems such as MLPQ. Those who are familiar with abstract interpretations with widening operators may skip the translations steps and consider an invariant analysis similar to abstract interpretation with the widening operator replaced by *l-u widening* or the non-convex approximation.

Example 8. Consider again the subway train example. The Datalog with addition constraint program that is equivalent to the subway train control program is described in [**?**]. Let $\mathcal{E}$, the error states, be as follows:

$$\mathcal{E} = \{b, s \ : \ |b - s| > 20\}.$$

It can be checked that the over-approximation found for the subway train and $\mathcal{E}$ have no common solution. Hence the subway train program is safe to use.

Example 9. Consider again the program in Example **??**. This program can be translated into the following constraint Datalog program.

$$\begin{array}{ll} Line2(x, y) & :\!-\!- \ x = 1, \ y = 1. \\ Line2(x', y') & :\!-\!- \ Line2(x, y), \ x' = x + 1, \ y' = y + 2x' - 1. \\ Line3(x, y) & :\!-\!- \ Line2(x, y), \ x' = x + 1, \ y' = y + 2x' - 1. \end{array}$$

We can calculate the over-approximation of the above Datalog program similar to Example **??**. It is interesting to see how the bottom up evaluation terminates. In the 11th application of the second rule (which corresponds to the 12th entry of line 2 in the original program), the bottom up evaluation finds that $\exists x, y \ x' = x + 1, \ y' = y + 2x' - 1, \ x \geq 11, \ y \geq 121 \ = \ x' \geq 12, y' \geq 144$. Before adding this to the set of already existing ABMs for Line2, we need to replace x' by x and y' by y. The replacement yields $x \geq 12, y \geq 144$, which is more restrictive than $x \geq 11, y \geq 121$, the previously added ABM. Hence it is not added to the set of ABMs for $Line2$ and the evaluation terminates.[3]

Let the error states $\mathcal{E}$ be as follows:

$$\mathcal{E} = \{y \ : \ y \geq 10, \ -y \geq -15\}.$$

This is a region where y is between 10 and 15 inclusively. It can be easily checked that the over-approximation of $L_{2,1}$ in Example **??** and $\mathcal{E}$ have no common solution. Hence the program can never enter the error states.

[3] This is a simplification of the bottom up evaluation, because within the ABMs constraints of the form $x \geq b$ are represented by $x^+ - x^- \geq 2b$ as described in Section **??**.

7 An Efficient Representation of ABMs

Without loss of generality we can fix an order of the variables and assume that in all addition constraints of the form $x - y \geq b$ or $-x + y \geq b$, x is earlier than y, and in all addition constraints of the form $x + y \geq b$ and $-x - y \geq b$ x is earlier than y or $x = y$. We can represent lower bound constraints of the form $x \geq b$ by $x + x \geq 2b$ and and upper bound constraints of the form $-x \geq b$ by $-x - x \geq 2b$.

Then we can represent $x - y \geq b$ and $x + y \geq b$ constraints by a matrix L as follows:

$$L[i,j] = \left\{ \begin{array}{ll} b & \text{if } (x_i - x_j \geq b) \in C \text{ and } i < j \\ b & \text{if } (x_j + x_i \geq b) \in C \text{ and } j \leq i \\ -\infty & \text{otherwise} \end{array} \right\}$$

Similarly, we can represent $-x + y \geq b$ and $-x - y \geq b$ constraints by a matrix U as follows:

$$U[i,j] = \left\{ \begin{array}{ll} b & \text{if } (-x_i + x_j \geq b) \in C \text{ and } i < j \\ b & \text{if } (-x_j - x_i \geq b) \in C \text{ and } j \leq i \\ -\infty & \text{otherwise} \end{array} \right\}$$

Note that the above is equivalent to the following:

$$U[i,j] = \left\{ \begin{array}{ll} b & \text{if } (x_i - x_j \leq -b) \in C \text{ and } i < j \\ b & \text{if } (x_j + x_i \leq -b) \in C \text{ and } j \leq i \\ -\infty & \text{otherwise} \end{array} \right\}$$

For example, the ABM in Example **??** can be represented by the following matrices. L is:

	x	y
x	$-\infty$	4
y	10	6

and U is:

	x	y
x	-50	$-\infty$
y	-40	$-\infty$

The above representation with matrices L and U requires only $2n^2$ matrix entries, while Miné's representation requires $4n^2$ matrix entries. Moreover, since the corresponding entries in L and U are lower and upper bounds of the same $x_i - x_j$ or $x_j + x_i$, they can be put together as follows:

	x	y
x	$[-\infty, 50]$	$[4, +\infty]$
y	$[10, 40]$	$[6, +\infty]$

Therefore, each ABM can be represented using a matrix with only n^2 entries that are intervals.

8 Related and Future Work

Pratt [**?**] gave efficient algorithms for testing the satisfiability and the implication problem for conjunctions of potential constraints. Harvey and Stuckey [**?**] gave a polynomial algorithm for the implication problem in the case of conjunctions of sum constraints with integer variables. An open problem is to improve the complexity of the algorithm in [**?**]. Currently, we are working on updating the MLPQ system to the more efficient ABM representation described in Section **??**. Another open problem is to find conditions when the over-approximation and the under-approximation of the program semantics are the same, resulting in a precise evaluation.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138*, 1 (1995), 3–34.
2. Anderson, S., and Revesz, P. Verifying the incorrectness of programs and automata. In *Proc. 6th International Symposium on Abstraction, Reformulation, and Approximation* (2005), vol. 3607 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1–13.
3. Clarke, E. M., Grumberg, O., and Peled, D. A. *Model Checking*. MIT Press, 1999.
4. Colmerauer, A. Note sur Prolog III. In *Proc. Séminaire Programmation en Logique* (1986), pp. 159–174.
5. Cousot, P. Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming. In *Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)* (Paris, France, LNCS 3385, Jan. 17–19 2005), Springer, Berlin, pp. 1–24.
6. Cousot, P., and Cousot, R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. ACM Principles on Programming Languages* (1977), ACM Press, pp. 238–252.
7. Cousot, P., and Halbwachs, N. Automatic discovery of linear restraints among variables of a program. In *Proc. ACM Principles on Programming Languages* (1978), ACM Press, pp. 84–97.
8. Delzanno, G., and Podelski, A. Model checking in CLP. In *2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (1999), vol. 1579 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 74–88.
9. Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T., and Berthier, F. The constraint logic programming language chip. In *Proc. Fifth Generation Computer Systems* (Tokyo, Japan, 1988), pp. 693–702.
10. Fribourg, L., and Olsén, H. A decompositional approach for computing least fixed-points of Datalog programs with Z-counters. *Constraints 2*, 3–4 (1997), 305–36.
11. Fribourg, L., and Richardson, J. D. C. Symbolic verification with gap-order constraints. In *Proc. Logic Program Synthesis and Transformation* (1996), vol. 1207 of *Lecture Notes in Computer Science*, pp. 20–37.
12. Godefroid, P., Huth, M., and Jagadeesan, R. Abstraction-based model checking using modal transition systems. In *12th International Conference on Concurrency Theory* (2001), pp. 426–440.
13. Halbwachs, N. Delay analysis in synchronous programs. In *Proc. Conference on Computer-Aided Verification* (1993), pp. 333–46.
14. Harvey, W., and Stuckey, P. A unit two variable per inequality integer constraint solver for constraint logic programming. In *Proc. Australian Computer Science Conference (Australian Computer Science Communications)* (1997), pp. 102–11.

15. Jaffar, J., and Lassez, J. L. Constraint logic programming. In *Proc. 14th ACM Symposium on Principles of Programming Languages* (1987), pp. 111–9.
16. Jaffar, J., and Maher, M. Constraint logic programming: A survey. *J. Logic Programming 19/20* (1994), 503–581.
17. Kanellakis, P. C., Kuper, G. M., and Revesz, P. Constraint query languages. In *Proc. ACM Symposium on Principles of Database Systems* (1990), pp. 299–313.
18. Kanellakis, P. C., Kuper, G. M., and Revesz, P. Constraint query languages. *Journal of Computer and System Sciences 51*, 1 (1995), 26–52.
19. Kerbrat, A. Reachable state space analysis of lotos specifications. In *Proc. 7th International Conference on Formal Description Techniques* (1994), pp. 161–76.
20. Kuper, G. M., Libkin, L., and Paredaens, J., Eds. *Constraint Databases*. Springer-Verlag, 2000.
21. Marriott, K., and Stuckey, P. J. *Programming with Constraints: An Introduction*. MIT Press, 1998.
22. McMillan, K. *Symbolic Model Checking*. Kluwer, 1993.
23. Miné, A. The octagon abstract domain. In *Proceedings Analysis, Slicing and Transformation* (2001), IEEE Press, pp. 310–319.
24. Pratt, V. Two easy theories whose combination is hard. *MIT Technical Report* (1977).
25. Revesz, P. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. *Theoretical Computer Science 116*, 1 (1993), 117–49.
26. Revesz, P. Datalog programs with difference constraints. In *Proc. 12th International Conference on Applications of Prolog* (1999), pp. 69–76.
27. Revesz, P. Reformulation and approximation in model checking. In *Proc. 4th International Symposium on Abstraction, Reformulation, and Approximation* (2000), B. Choueiry and T. Walsh, Eds., vol. 1864 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 124–43.
28. Revesz, P. *Introduction to Constraint Databases*. Springer-Verlag, 2002.
29. Revesz, P., Chen, R., Kanjamala, P., Li, Y., Liu, Y., and Wang, Y. The MLPQ/GIS constraint database system. In *ACM SIGMOD International Conference on Management of Data* (2000).