Solving Difference Constraints over Modular Arithmetic

Graeme Gange, Harald Søndergaard, Peter J. Stuckey, and Peter Schachte

Department of Computing and Information Systems,
The University of Melbourne, Victoria 3010, Australia
{ggange,harald,pjs,schachte}@csse.unimelb.edu.au

Abstract. Difference logic is commonly used in program verification and analysis. In the context of fixed-precision integers, as used in assembly languages for example, the use of classical difference logic is unsound. We study the problem of deciding difference constraints in the context of modular arithmetic and show that it is strongly NP-complete. We discuss the applicability of the Bellman-Ford algorithm and related shortest-distance algorithms to the context of modular arithmetic. We explore two approaches, namely a complete method implemented using SMT technology and an incomplete fixpoint-based method, and the two are experimentally evaluated. The incomplete method performs considerably faster while maintaining acceptable accuracy on a range of instances.

1 Introduction

We consider the problem of adapting classical difference logic over $\mathbb{Z}$ [2] to the congruence class used in modulo-m integer arithmetic, here denoted $\mathbb{Z}_m$. Understanding this class is important for the design of automated reasoning that is concerned with machine arithmetic. Our particular interest in this arises from our work on analysis and verification of low-level code. We wish to improve static analysis techniques for low-level programming languages that use w-bit fixed-precision integers, that is, we are interested in the particular case $m = 2^w$. Much of the literature on program analysis and software verification uses difference logic and similar numeric abstract domains, tacitly assuming *unbounded* integers. It is well known that, in that context, difference logic can be decided in $O(|V||C|)$ deterministic time, for variables V and constraints C. In the context of $\mathbb{Z}_m$, the decision problem becomes strongly NP-complete.

Consider the program fragment shown in Figure 1. Conventional static analysis with difference bound matrices [4] or octagons [11] will derive the bounded difference constraint $0 \leqslant y - x \leqslant 6$ and determine that the branch $y < x$ will never be executed. In a context of fixed-precision integers, owing to possible overflow, this conclusion is clearly wrong. Nevertheless, in some sense the derived invariant is meaningful, as y lies between x and $x + 6$ on the modular integer number circle.

The challenge is to ensure that program analysis "understands" machine operations so as to remain faithful to machine arithmetic. Previous work has

```
unsigned int x = ⋆;
unsigned int y = x;
for(int i = 0; i < 6; i ++)
    if(⋆)
        y ++;
if(y < x) ERROR;
```

Fig. 1: Unbounded relational analysis will deem **ERROR** unreachable; however, on systems with fixed-width integers, y may wrap to 0 if x is very large.

mainly dealt with non-relational abstract domains, notably *interval* domains [15, 16, 8]. Simon and King [17] considered adapting convex polyhedra to modular arithmetic by computing a convex approximation relative to a fixed wrapping point. Other approaches consider instead systems of equations [13] and disequations [9] under modular arithmetic. $\mathbf{SMT}(\mathcal{BV})$ [10] problems involve a variety of constraints over $\mathbb{Z}_{2^w}$; solvers for these problems typically convert the arithmetic operations to SAT. These techniques are complete, but may be too slow to be viable for certain applications, such as invariant synthesis.

One critical issue with classical abstract domains such as interval domains, octagons, and so on, is that they rely on having a linear ordering, $\leqslant$, on the set of integers. The only way to capture a non-trivial concept of ordering on $\mathbb{Z}_m$ is to discuss order only with respect to some reference point. For example, we may decide that $x \leqslant y$ means, loosely, that "starting from 0 and moving clockwise on the number circle, x is encountered no later than y"—a natural reading when unsigned integer representation is used. Or, we may decide that it means "starting from $-m/2$, x is met no later than y"—when signed representation is used. To complicate matters, many low-level languages, such as LLVM and assembly languages, fail to provide signedness information, relying on the fact that arithmetic operations such as addition, subtraction and multiplication are agnostic with respect to signedness. Navas *et al.* [14] point out that analysis of such languages, in order to maintain precision, has to be signedness-agnostic as well, which means superposing signed/unsigned assumptions during analysis.

For the remainder of this paper, we assume all inequalities are unsigned. Signed inequalities $x \leqslant_{\mathrm{s}} y$ can be expressed in terms of unsigned inequalities: $x \leqslant_{\mathrm{s}} y$ iff $x + \frac{m}{2} \leqslant y + \frac{m}{2}$. This does, however, require the introduction of shifted variables $x' = (x + \frac{m}{2}) \bmod m$ and $y' = (y + \frac{m}{2}) \bmod m$.

Classical integer difference constraints provide lower and upper bounds on integer differences $x - y$, and these bounds have consequences for order. For example, for positive k, a constraint $x - y \geqslant k$ allows us to deduce $x \geqslant y$. When we move to $\mathbb{Z}_m$, this link between difference and order is lost. For example, assuming signed arithmetic, $[x \mapsto -2^{w-1}, y \mapsto 2^{w-1} - 1]$ satisfies $x - y \geqslant 0$ but not $x \geqslant y$. Hence an important step towards getting a handle on "wrapped" difference logic is to separate the aspects of proximity and (relative) order.

The following contributions are made in this paper:

```
bellman_ford(⟨V, E⟩)
    % Introduce a fresh least element.
    V' = V ∪ {v'}
    E' = E ∪ {⟨v', 0, v_i⟩ | v_i ∈ V}
    % Initialize relations
    D(v') := 0
    for(v_i ∈ V)
        D(v_i) := −∞
    % Progressively expand the set of paths to each node.
    for(k ∈ {1, ..., |V|})
        for((⟨v_i, w, v_j⟩ ∈ E')
            D(v_j) = max(D(v_j), D(v_i) + w)
    % Check for any inconsistencies
    for((⟨v_i, w, v_j⟩ ∈ E')
        if(D(v_i) + w > D(v_j))
            return UNSAT
    return SAT
```

Fig. 2: Pseudo-code for the Bellman-Ford algorithm for checking satisfiability of a set of unbounded difference logic constraints.

- We study the complications that arise when reasoning about difference constraints takes place in the presence of modular arithmetic.
- We offer a simple proof that, in that context, for $m > 2$, decidability of difference constraints is NP-complete.
- We propose a framework for combined reasoning about proximity and order and use this to develop an efficient but incomplete decision procedure.
- We evaluate the resulting method, comparing it to two more traditional SMT-based decision procedures.

In Section 2 we recapitulate the classical case. In Section 3 we discuss "wrapped" difference constraints and develop the different approaches: a complete method based on $\mathbf{SMT}(\mathcal{BV})$ (bit-vector) technology, one using $\mathbf{SMT}(\mathcal{DL})$ (difference logic), and an incomplete fixpoint-based method. Section 4 contains the evaluation and Section 5 concludes.

2 Deciding difference logic

The classical method for deciding difference logic[1] is the Bellman-Ford algorithm [2]. For later reference, we show it in Figure 2. It uses a graph representation of constraints, each variable giving rise to a node, and each difference

[1] The term "separation logic" is sometimes used [18, 19]. To avoid confusion with Reynolds-O'Hearn separation logic, we use the alternative "difference logic".

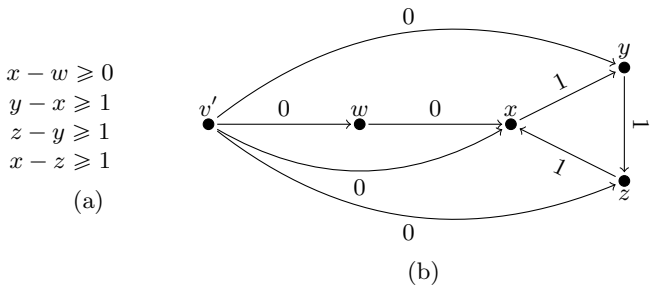


Fig. 3: (a) A set of difference constraints. (b) the corresponding graph representation (v' is a freshly introduced root vertex).

constraint giving rise to a weighted directed edge. The algorithm relies on these inference rules for difference logic ($\bot$ denotes unsatisfiability):

Inversion:	$\alpha \leqslant y - x \leqslant \beta$ iff $-\beta \leqslant x - y \leqslant -\alpha$
Resolution:	$\dfrac{\alpha_1 \leqslant y - x \leqslant \beta_1 \quad \alpha_2 \leqslant z - y \leqslant \beta_2}{\alpha_1 + \alpha_2 \leqslant z - x \leqslant \beta_1 + \beta_2}$
Tightening:	$\dfrac{\alpha_1 \leqslant y - x \leqslant \beta_1 \quad \alpha_2 \leqslant y - x \leqslant \beta_2}{\mathbf{max}(\alpha_1, \alpha_2) \leqslant y - x \leqslant \mathbf{min}(\beta_1, \beta_2)}$
Contradiction:	$\dfrac{\alpha \leqslant y - x \leqslant \beta, \alpha > \beta}{\bot}$

Example 1. Consider the set of constraints shown in Figure 3(a). The graph corresponding to these constraints is given in Figure 3(b). Note that the constraints are satisfiable if and only if there are no positive-weight cycles in the graph.[2] After computing the longest paths of length up to $|V|$, we have:

$$\{D(v') = 0, D(w) = 0, D(x) = 3, D(y) = 3, D(z) = 3\}$$

However, performing another iteration would still increase the path lengths, since, for example, $D(z) + 1 > D(x)$. This indicates the presence of a positive-weight cycle. □

3 Wrapped difference constraints

In this section, we consider systems of constraints of the form $\alpha \in y - x \in \beta$ (which we sometimes write it as $y - x \in [\alpha, \beta]$) under modular arithmetic. Given that

[2] Presentations of difference logic sometimes express the problem in terms of shortest (rather than longest) paths, in which case unsatisfiability corresponds to negative (rather than positive) cycles.

we are concerned with numerical *proximity* as well as ordering, we allow these intervals to cross m; such a *wrapped* interval $[\alpha, \beta]$ is interpreted as follows:[3]

$$\gamma[\alpha,\beta] = \begin{cases} \{d \mid \alpha \leqslant d \wedge d \leqslant \beta\} & \textbf{if } \alpha \leqslant \beta \\ \{d \mid \alpha \leqslant d \wedge d \leqslant m-1\} \cup \{d \mid 0 \leqslant d \wedge d \leqslant \beta\} & \textbf{otherwise} \end{cases}$$

For example, in a modulo-16 context, the wrapped interval $[14, 2]$ denotes the set $\{0, 1, 2, 14, 15\}$. In this modulo-m context, a one-sided constraint (such as $y - x \geqslant 3$) is essentially meaningless; the usual inequalities under $\mathbb{Z}_m$ are implicitly bounded by 0 and $m - 1$. On the other hand, containment of wrapped intervals is easily expressed:

$$[\alpha,\beta] \sqsubseteq [\alpha',\beta'] \text{ iff } \gamma[\alpha,\beta] \subseteq \gamma[\alpha',\beta']$$

We can perform certain operations, similar to those of Section 2, on the number circle. However, of Section 2's inference rules, only inversion remains sound.

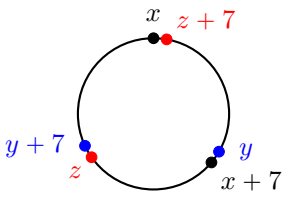
Example 2. Consider, for $k > 1$, this set of constraints:

$$1 \leqslant y - x \leqslant k, \quad 1 \leqslant z - y \leqslant k, \quad 1 \leqslant x - z \leqslant k \tag{1}$$

Resolving the first two constraints, we get $2 \leqslant z - x \leqslant 2k$, or, equivalently, $-2k \leqslant x - z \leqslant -2$. Under standard difference logic, (1) would be unsatisfiable:

$$\dfrac{\dfrac{1 \leqslant x - z \leqslant k \qquad -2k \leqslant x - z \leqslant -2}{1 \leqslant x - z \leqslant -2}}{\bot}$$

However, as illustrated here, in the modular-arithmetic case, this set of constraints is satisfiable if k is sufficiently large. For example, the constraints are satisfiable in $\mathbb{Z}_{16}$, for $k = 7$ (take, say, $x = 1, y = 6, z = 11$). □



The n constraints

$$\alpha_1 \leqslant x_2 - x_1 \leqslant \beta_1, \ldots, \alpha_{n-1} \leqslant x_n - x_{n-1} \leqslant \beta_{n-1}, \alpha_n \leqslant x_1 - x_n \leqslant \beta_n$$

induce the constraint $\alpha \leqslant 0 \leqslant \beta$, where $\alpha = \alpha_1 + \cdots + \alpha_n$ and $\beta = \beta_1 + \cdots + \beta_n$. This latter constraint is unsatisfiable exactly when 0 falls outside $[\alpha, \beta]$, a condition which is reminiscent of the positive-weight cycle condition for conventional difference logic.

In principle a variant of a shortest-path algorithm could be used to detect these inconsistent cycles; however, this requires computing the intersection of

[3] The definition overloads the square bracket notation: The function γ takes a possibly *wrapped* interval and expresses its meaning in terms of ordinary intervals.

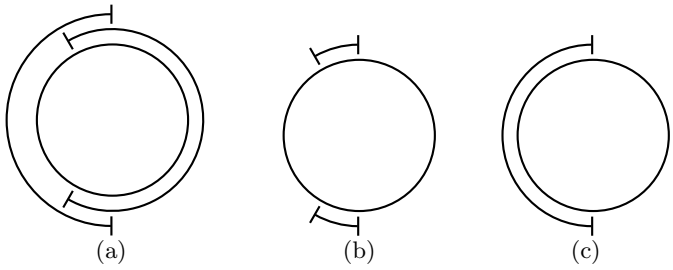


Fig. 4: (a) A pair of intervals on the number circle, (b) the intersection of the two intervals and (c) an optimal over-approximation of the intersection.

wrapped intervals. Consider the two intervals shown in Figure 4(a). The intersection of these two intervals is no longer a single interval. Indeed, the intersection of k wrapped intervals produces up to $min(k, \frac{m}{2})$ disjoint feasible intervals; when combined with resolution, $min(2^{\frac{k}{2}}, \frac{m}{2})$ intervals can be generated. For example, with these constraints:

$$0 \leqslant y - x \leqslant 2,\ 0 \leqslant z - y \leqslant 4,\ 2 \leqslant y - x \leqslant m,\ 4 \leqslant z - y \leqslant m$$

we have $y - x \in \{0, 2\}$ and $z - y \in \{0, 4\}$, which yields $z - x \in \{0, 2, 4, 6\}$. There are four equally good interval approximations of this set.

3.1 Interval sets

We could represent the feasible relations between two variables exactly by explicitly maintaining the set of (disjoint) feasible intervals. To reason about bounded difference constraints, we require two operations: intersection of interval-sets (denoted $\sqcap$) and pointwise addition of interval-sets (denoted $+$).

$$A \sqcap A' = \bigsqcup_{[\alpha,\beta]\in A} \bigsqcup_{[\alpha',\beta']\in A'} \left\{ \begin{array}{ll} [\alpha, \beta] & \textbf{if } \alpha \in [\alpha', \beta'] \wedge \beta \in [\alpha', \beta'] \\ [\alpha, \beta'] & \textbf{if } \alpha \in [\alpha', \beta'] \wedge \beta' \in [\alpha, \beta] \\ [\alpha', \beta] & \textbf{if } \alpha' \in [\alpha, \beta] \wedge \beta \in [\alpha', \beta'] \\ [\alpha', \beta'] & \textbf{if } \alpha' \in [\alpha, \beta] \wedge \beta' \in [\alpha, \beta] \end{array} \right\}$$

$$A + A' = \bigsqcup_{[\alpha,\beta]\in A} \bigsqcup_{[\alpha',\beta']\in A'} \{[\alpha + \alpha', \beta + \beta']\}$$

The operation $\sqcap$ can be implemented in $O(|A| + |A'|)$ time; $+$ requires $O(|A||A'|)$ time in the worst case. In the case of $+$, the results are normalized by merging overlapping intervals. Note that $+$ and $\sqcap$ are both commutative and associative. Also note that the full interval, $\top$, is a neutral element for $\sqcap$, while $\{[0, 0]\}$ is neutral for $+$. It would be convenient if the resulting structure formed a semiring, as this would allow us to use the algebraic shortest distance framework of Mohri [12]. Unfortunately, while we do have the property

$$(a \sqcap b) + c \sqsubseteq (a + c) \sqcap (b + c)$$

it is *not* the case that + distributes over $\sqcap$. As a counter-example, consider $\mathbb{Z}_{16}$, and take $A = [0, 8]$ and $B = [9, 0]$. We have $A + (A \sqcap B) = A + [0, 0] = A$, while $(A + A) \sqcap (A + B) = \top \sqcap \top = \top$. As we shall see in Section 3.6, this complicates the operation of longest path algorithms.

3.2 Wrapped-interval approximation

For the analysis of programs that use $\mathbb{Z}_m$ for a large m (and usually m is 2^{32} or 2^{64}), representing the feasible values precisely is impractical. In this section, we propose the construction of an over-approximation of the set of feasible intervals.

We adopt a "wrapped interval" representation [14], approximating the set of feasible intervals with a single interval. A wrapped interval is any sequence of consecutive numbers on the modulo-m number circle; for example, with $m = 16$, the interval $[8, 0]$ is the set $\{8, 9, \ldots, 15, 0\}$. Given a set of integers modulo m, there may be several minimal approximations in the form of wrapped intervals; for example, the set $\{0, 8\}$ may be approximated by $[0, 8]$ or by $[8, 0]$, two intervals of equal cardinality. To ensure a deterministic choice, we use a total ordering $\preceq$ over wrapped intervals, ordering them primarily by cardinality and then lexicographically (we write $x \ \oplus_m \ y$ for $(x \oplus y) \bmod m$ for binary infix operator $\oplus$):

$$[\alpha, \beta] \preceq [\alpha', \beta'] \textbf{ iff } (\beta -_m \alpha) < (\beta' -_m \alpha') \vee ((\beta -_m \alpha = \beta' -_m \alpha') \wedge \alpha \leqslant \alpha')$$

This allows us to define an over-approximation of the meet which selects the approximation with minimum cardinality, breaking ties by favouring the lexicographically smallest left-bound.

$$[\alpha, \beta] \sqcap_{\mathbb{L}} [\alpha', \beta'] = \begin{cases} \textbf{if} & \alpha \notin [\alpha', \beta'] \wedge \alpha' \notin [\alpha, \beta] & \textbf{then} & \bot \\ \textbf{else if} & [\alpha, \beta] \sqsubseteq [\alpha', \beta'] & \textbf{then} & [\alpha, \beta] \\ \textbf{else if} & [\alpha', \beta'] \sqsubseteq [\alpha, \beta] & \textbf{then} & [\alpha', \beta'] \\ \textbf{else if} & \alpha' \notin [\alpha, \beta] & \textbf{then} & [\alpha, \beta'] \\ \textbf{else if} & \alpha \notin [\alpha', \beta'] & \textbf{then} & [\alpha', \beta] \\ \textbf{else} & \textbf{min}_{\preceq}([\alpha, \beta], [\alpha', \beta']) & & \end{cases}$$

$$[\alpha, \beta] + [\alpha', \beta'] = \begin{cases} \top & \textbf{if } (\beta -_m \alpha) + (\beta' -_m \alpha') \geqslant m - 1 \\ [\alpha +_m \alpha', \beta +_m \beta'] & \textbf{otherwise} \end{cases}$$

Notice that, using this approximation, $\sqcap_{\mathbb{L}}$ lacks several properties provided by typical lattice operations. $\sqcap_{\mathbb{L}}$ is absorptive and commutative but not associative.

Example 3. With $m = 16$, consider $A = [8, 15]$, $B = [12, 9]$, and $C = [0, 10]$. We have $(A \sqcap_{\mathbb{L}} B) \sqcap_{\mathbb{L}} C = [8, 15] \sqcap_{\mathbb{L}} [0, 10] = [8, 10]$. On the other hand, we have $A \sqcap_{\mathbb{L}} (B \sqcap_{\mathbb{L}} C) = [8, 15] \sqcap_{\mathbb{L}} [0, 9] = [8, 9]$. □

Also, $\sqcap_{\mathbb{L}}$ is not monotone (nor decreasing) with respect to the inclusion ordering $\sqsubseteq$ (however, $\sqcap_{\mathbb{L}}$ *is* monotone with respect to the cardinality ordering $\preceq$).

Example 4. Consider the intervals $A = [0, 10]$, $A' = [0, 8]$, $B = [8, 1]$ on $\mathbb{Z}_{16}$. A' is clearly a subset of A, however $A \sqcap_{\mathbb{L}} B$ is incomparable with $A' \sqcap_{\mathbb{L}} B$. Namely, $A \sqcap_{\mathbb{L}} B = [8, 1]$, while $A' \sqcap_{\mathbb{L}} B = [0, 8]$. So $\sqcap_{\mathbb{L}}$ fails to be monotone. □

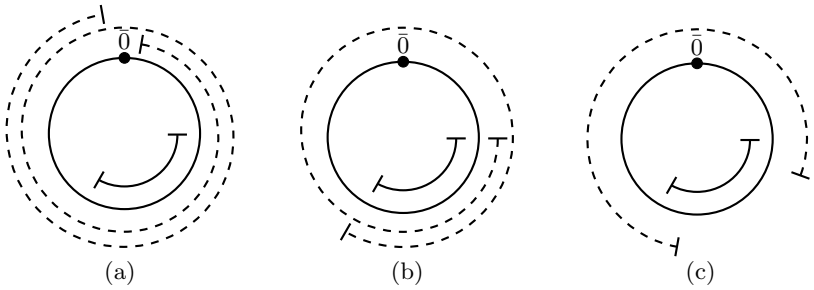


Fig. 5: (a) When the concrete range covers the entire circumference of the number circle, the proximity bounds cannot be reduced further. (b) The concrete bounds can, however, be reduced to the next corresponding proximity bound. (c) If both sets of endpoints are mutually contained, there can be no reduction.

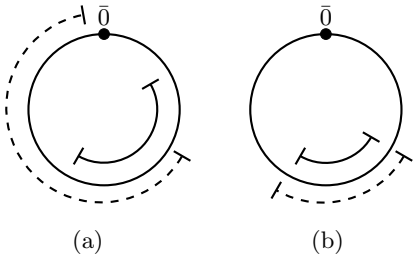


Fig. 6: If the union of the concrete and proximity intervals do not cover the entire number circle, each can be tightened to the region satisfying both.

3.3 Combining wrapped difference with relative order

The wrapped interval constraints discussed so far express proximity only. They cannot express constraints such as $x < y$. This can be fixed, however, by allowing "concrete" interval information. Thus we combine proximity and range constraints in pairs $\langle [\alpha, \beta], [d, D] \rangle$ with the semantics:

$$\gamma\langle [\alpha, \beta], [d, D] \rangle = \{(x, y) \in \mathbb{Z}_m^2 \mid y -_m x \in [\alpha, \beta] \wedge y - x \in [d, D]\}$$

Note that, with $x, y \in [0, m-1]$, the value of $y - x$ can be anywhere between $-m+1$ and $m-1$ ($2m-1$ possible values). Hence $\top = \langle [0, m-1], [-m+1, m-1] \rangle$. Figure 5(a) depicts an interval pair (assuming $m = 16$, the pair is $\langle [4, 9], [-15, 15] \rangle$), the first interval shown by a solid arc, the second by a dashed arc.

We normalise interval pairs by propagating information from each component to the other. Let d_m be d projected onto the range $[0, m-1]$. We can use this to determine how far the lower (respectively upper) bound of the concrete range must be adjusted to reach the corresponding proximity bound. This difference is then mapped back onto the concrete range. Note the use of $[d, D]_m$, the

projection of the interval onto $[0, m-1]$, defined as $[d, D]_m = [0, m-1]$ if $D - d \geqslant m$, and $[d, D]_m = [d_m, D_m]$, otherwise.

$$\begin{aligned}
&\textbf{norm}\ \langle [\alpha, \beta], [d, D] \rangle = \langle [\alpha', \beta'], [d', D'] \rangle \quad \textbf{where} \\
&d' = \begin{cases} d & \textbf{if } d_m \in [\alpha, \beta] \\ d + (\alpha -_m d_m) & \textbf{otherwise} \end{cases} \\
&D' = \begin{cases} D & \textbf{if } D_m \in [\alpha, \beta] \\ D - (D_m -_m \beta) & \textbf{otherwise} \end{cases} \\
&[\alpha', \beta'] = \begin{cases} [\alpha, \beta] & \textbf{if } \alpha \in [d, D]_m \wedge d_m \in [\alpha, \beta] \\ [\alpha, \beta] \sqcap [d, D]_m & \textbf{otherwise} \end{cases}
\end{aligned}$$

Details of the definition are justified by considering the cases shown in Figures 5 and 6. The following theorem says that **norm** establishes the tightest possible consistent bounds.

Theorem 1. Let $\langle A', C' \rangle = \textbf{norm}\ \langle A, C \rangle$. For all A'', C'', if $\gamma\langle A'', C'' \rangle = \gamma\langle A, C \rangle$ then $A' \sqsubseteq A'' \wedge C' \sqsubseteq C''$.

Proof. Consider a pair $\langle [\alpha', \beta'], [d', D'] \rangle = \textbf{norm}\ \langle [\alpha, \beta], [d, D] \rangle$. In the case that $\alpha \in [d_m, D_m]$ and $d_m \in [\alpha, \beta]$, as illustrated in Figure 5(c), neither bound can be tightened. If $[\alpha, \beta]$ and $[d, D]_m$ do not intersect, we have $d + (\alpha -_m d_m) > D$, and the result correctly represents $\bot$.

This leaves the case where there is some overlap between $[\alpha, \beta]$ and $[d, D]_m$, but the intervals do not cross at both ends. In that case, $[\alpha, \beta] \sqcap [d, D]_m$ is the largest interval consistent with both $[\alpha, \beta]$ and $[d, D]$. If $d_m \notin [\alpha, \beta]$, we must adjust d to the next point on the number circle consistent with $[\alpha, \beta]$ – that is, α. The minimum distance d must be shifted is $\alpha -_m d_m$, in which case $d'_m = \alpha$. By similar reasoning, if $D_m \notin [\alpha, \beta]$, we must reduce D by $D_m -_m \beta$, giving $D'_m = \beta$. Both d'_m and D'_m are in $[\alpha', \beta']$. Assume there were some element of $c \in [\alpha', \beta']$ such that $c \notin [d', D']_m$. As $c \in [\alpha', \beta']$, we have $c \in [\alpha, \beta] \wedge c \in [d, D]$. Then c is either in the interval $[d, d')$, or $(D', D]$. However, this cannot be the case, as there are no elements of $[\alpha, \beta]$ in either interval. Therefore, all elements of $[\alpha', \beta']$ must be consistent with $[d', D']$.

We conclude that **norm** computes the tightest intervals that preserve the semantics of the input pair. □

In the case of the complete interval set representation, $[\alpha', \beta']$ can be computed with a standard join; the additional case is to avoid losing information when the intervals overlap in two places (analogous to the case in Figure 4(a)). Using normalisation, we can define the necessary operators on the combined domain:

$$\begin{aligned}
&\langle [\alpha_x, \beta_x], [d_x, D_x] \rangle \sqcap \langle [\alpha_y, \beta_y], [d_y, D_y] \rangle = \\
&\qquad \textbf{norm}\ \langle [\alpha_x, \beta_x] \sqcap [\alpha_y, \beta_y], [\textbf{max}(d_x, d_y), \textbf{min}(D_x, D_y)] \rangle
\end{aligned}$$

$$\begin{aligned}
&\langle [\alpha_x, \beta_x], [d_x, D_x] \rangle + \langle [\alpha_y, \beta_y], [d_y, D_y] \rangle = \\
&\qquad \textbf{norm}\ \langle [\alpha_x, \beta_x] + [\alpha_y, \beta_y], [d_x + d_y, D_x + D_y] \rangle
\end{aligned}$$

3.4 NP-completeness of wrapped difference constraints

As already observed by Bjørner *et al.* [1], wrapped difference constraints are NP-complete. In this section we give a simpler proof, using, as do Bjørner *et al.*, reduction from graph 3-colourability. We strengthen the result [1] by showing NP-completeness for all cases $m > 2$. For $m = 2$, the problem can be solved in polynomial time in the same manner as 2-colouring.

First, given an assignment to variables $\{v_1, \ldots, v_n\}$, we can check, in polynomial time, whether each difference constraint is satisfied; so wrapped difference logic is in NP. It remains to show that the problem is NP-hard.

Assume $m > 2$; consider a 3-COLOURABILITY instance $G = \langle \{v_1, \ldots, v_n\}, E \rangle$. We construct a system of $|E| + n$ difference constraints in $\mathbb{Z}_m$ over variables $\{x, x_1, \ldots, x_n\}$:

- For each vertex v_i, introduce the constraint $x_i - x \in [0, 2]$ (for $m = 3$ this is a vacuous constraint, so it can be omitted).
- For each edge $(v_i, v_j) \in E$, introduce the constraint $x_i - x_j \in [1, m - 1]$.

The system of constraints can be generated in linear time. We claim that it is satisfiable iff G is 3-colourable.

Assume the set of constraints can be satisfied and let ν be a satisfying valuation. For each x_i, we have $\nu(x_i) \in \{\nu(x), \nu(x) +_m 1, \nu(x) +_m 2\}$, owing to the constraint $x_i - x \in [0, 2]$. Taking $\nu(x)$, $\nu(x) +_m 1$, and $\nu(x) +_m 2$ as three "colours", we choose the colour $\nu(x_i)$ for node v_i. This gives a 3-colouring of G, because, for adjacent vertices v_i and v_j, the colours $\nu(x_i)$ and $\nu(x_j)$ must be different, owing to the constraint $x_i - x_j \in [1, m - 1]$.

Conversely, assume that G is 3-colourable. Call the three colours used in the colouring 0, 1, and 2. We claim that the valuation ν which maps x to 0 and x_i to the colour of v_i satisfies the generated constraints. The constraints of form $x_i - x \in [0, 2]$ are satisfied by construction. The constraints of form $x_i - x_j \in [1, m - 1]$ are similarly satisfied, as the "difference" between the "colours" of v_i and v_j are precluded from being 0.

It follows that wrapped difference logic is NP-complete. Note that the usual directionality of edges in the graph generated by difference constraints is irrelevant here, since in modulo m arithmetic, $x - y \in [1, m - 1]$ and $y - x \in [1, m - 1]$ (and $x \neq y$) are equivalent. Also note that the reduction does not synthesize m from a 3-COLOURABILITY instance. Rather, m is a fixed constant in the transformation. As 3-COLOURABILITY is strongly NP-complete, and the transformation is pseudo-polynomial [7], wrapped difference logic is also strongly NP-complete.

3.5 SMT encodings: Two complete decision procedures

A common approach for solving problems over $\mathbb{Z}_{2^w}$ is *satisfiability modulo bit-vectors* ($\mathbf{SMT}(\mathcal{BV})$) [10]. In an $\mathbf{SMT}(\mathcal{BV})$ solver, each w-bit word x is typically translated into a vector v_x of w Boolean variables. Operations on $\mathbb{Z}_{2^w}$ are encoded using Boolean formulae to simulate the corresponding hardware circuit.

We can readily couch wrapped difference constraints in terms of $\mathbf{SMT}(\mathcal{BV})$. Letting $\texttt{-}_{\texttt{bv}}$ denote w-bit bit-vector subtraction, encode each constraint directly:

For a constraint $x \leqslant y$:	$v_x \leqslant_{\mathrm{u}} v_y$
For a constraint $y - x \in [i, j]$:	$(v_y \texttt{ -bv } v_x) \texttt{ -bv } i \leqslant_{\mathrm{u}} j \texttt{ -bv } i$

SMT($\mathcal{BV}$) solvers typically use complete methods for solving bit-vector constraints. These, then, provide a complete decision procedure for wrapped difference constraints.

An alternative way is to use *satisfiability modulo difference logic* (**SMT**($\mathcal{DL}$)). Each variable is constrained to the interval $[zero, zero + m - 1]$. We encode the concretization of a wrapped interval $[i, j]$ as a disjunction of concrete difference constraints, using similar reasoning to that illustrated in Figure 5:

For a variable x:	$0 \leqslant v_x - zero \leqslant m - 1$
For a constraint $x \leqslant y$:	$v_x \leqslant_{\mathrm{u}} v_y$
For a constraint $y - x \in [i, j]$:	$\begin{cases} \begin{pmatrix} -m + 1 \leqslant v_y - v_x \leqslant -m + j \\ \vee\ -m + i \leqslant v_y - v_x \leqslant j \\ \vee\ i \leqslant v_y - v_x \leqslant m - 1 \end{pmatrix} & \textbf{if } j_m < i_m \\ \begin{pmatrix} -m + i \leqslant v_y - v_x \leqslant -m + j \\ \vee\ i \leqslant v_y - v_x \leqslant j \end{pmatrix} & \textbf{otherwise} \end{cases}$

3.6 An incomplete decision procedure

Ideally, we would like an efficient, sound and complete decision procedure. Given that wrapped difference constraints are NP-complete, it seems highly unlikely that such a procedure exists. The SMT approaches are sound and complete, but can exhibit exponential running time. For use in an abstract interpretation framework, we require the analysis to be efficient, and we can afford to sacrifice completeness (but not soundness). We must therefore develop a sound over-approximation which maintains reasonable accuracy without excessive cost.

Given the similarities between wrapped difference constraints and classical difference logic, it seems likely that variants of shortest-path algorithms would provide suitable heuristics. Indeed, the problem of detecting a set of inconsistent wrapped difference constraints is very similar to the algebraic shortest distance framework of Mohri [12]. As observed in Section 3.1, our $\sqcap$ and $+$ operators lack some critical semiring properties; however, the structure of the problem remains the same. It is worth noting that all edges have an inverse; for any edge $y - x \in [\alpha, \beta]$, there is a corresponding edge $x - y \in [m - \beta, m - \alpha]$. As the inverse is easily computed, there is no need to store both edges explicitly. Similarly, we do not need to compute $(z-y)+(y-x)$ if we have already computed $(x - y) + (y - z)$.

In principle, the Bellman-Ford algorithm [2] provides a suitable sound over-approximation. Unfortunately, in the context of modular arithmetic, it quickly loses information about infeasible paths.

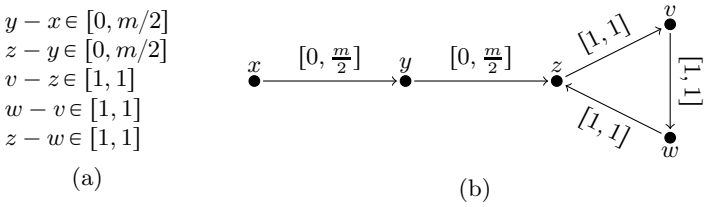


Fig. 7: The cycle $z \to w \to v \to z$ is inconsistent. However, this information is lost when applying Bellman-Ford from the root x.

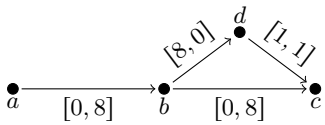


Fig. 8: After running the Floyd-Warshall algorithm (with variables ordered lexicographically), we have $c - a \in \top$, rather than the tightest bound of $[0, 9]$.

Example 5. Consider the set of constraints in Figure 7. The cycle $z \to w \to v \to z$ has value $[3, 3]$, which is an inconsistent self-loop (assuming $m > 3$). However, applying Bellman-Ford from root node x, we quickly determine that $z - x \in \top$. Then, as $\top + [1, 1] = \top$, we derive the same relation for $v - x$ and $w - x$. We cannot then deduce the existence of an inconsistent cycle. □

This example suggests that a single-source approach is unlikely to work. An alternative approach is to use an all-pairs shortest path algorithm to derive the strongest relation between each pair of variables. The obvious algorithm to use is the Floyd-Warshall algorithm [5]. However, as mentioned in Section 3.1, $+$ does not distribute over $\sqcap$; as a result, a direct application of the Floyd-Warshall algorithm is not guaranteed to reach a fixpoint with respect to every pair of variables.

Example 6. Consider a problem in $\mathbb{Z}_{16}$, with the constraint graph given in Figure 8. The algorithm computes the longest paths via first a then b (neither tightening any constraints). Since $d - c \in [15, 15]$ and $c - b \in [0, 8]$, paths via c tighten $d - b$ to $[8, 0] \sqcap [15, 7] = [15, 0]$ (note that $c - b$ is still $[0, 8]$, and $d - a$ is still $\top$). Once we compute paths via d, we tighten $c - b$ to $[0, 1]$. After the algorithm has finished, $c - a$ has still not yet been tightened to the correct value of $[0, 9]$. □

We could modify the Floyd-Warshall algorithm to continue iterating until a fixpoint is reached. However, this performs a great deal of redundant work; particularly given that most such constraint systems are quite sparse, so many edges are $\top$. Instead, we use a worklist-based algorithm to compute the fixpoint directly. We maintain a queue of (v_i, v_j) pairs which have changed, and update

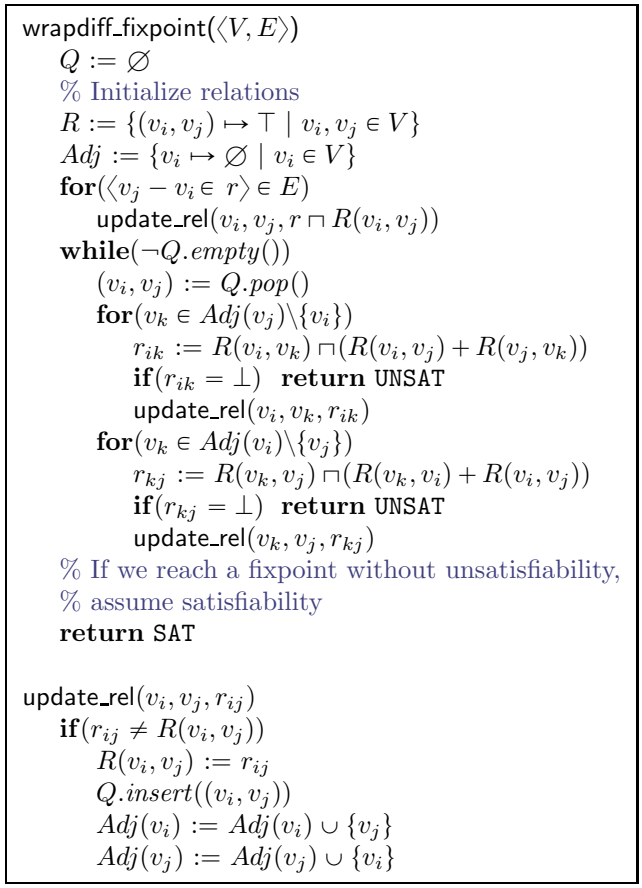


Fig. 9: Computation of a fixpoint of a set of difference constraints.

any adjacent (v_i, v_k) or (v_k, v_j) edges. This method is given in Figure 9. R maintains the relations between each pair of variables, and Q is the queue of updated edges. Since $c + \top = \top$ for all c, we need not compute $R(v_i, v_j) + R(v_j, v_k)$ unless both $R(v_i, v_j)$ and $R(v_j, v_k)$ are not $\top$. Adj holds, for each vertex v_i, the set of adjacent vertices v_k such that $R(v_i, v_k) \neq \top$. When an edge (v_i, v_j) is changed, we need only test elements in $Adj(v_i)$ and $Adj(v_j)$. We must, however, ensure Adj is updated whenever an edge ceases to be $\top$ – this is done in update_rel. Adj can be maintained in constant time with $O(n^2)$ space and initialization time. As mentioned, we do not need to keep track of both an edge and its inverse; we similarly avoid adding (v_j, v_i) to the queue if (v_i, v_j) has already been added.

This procedure is sound, as each step in the algorithm is the application of an inference of the form $x - z \sqsubseteq (x - y) + (y - z)$.

Theorem 2. The procedure wrapdiff_fixpoint terminates.

Proof. Using either the disjoint-set lattice or the interval over-approximation, the $\sqcap$ operation is monotone (according to the $\sqsubseteq$ and $\preceq$ orderings respectively) and non-increasing. At each step of wrapdiff_fixpoint, either some $R(v_i, v_k)$ or $R(v_k, v_j)$ must decrease, or all entries remain constant and the size of Q decreases. As both the disjoint-set and interval domains are finite, there cannot be any infinite descending chains. Hence wrapdiff_fixpoint must terminate. □

The fixpoint time complexity is clearly bounded by $\mathrm{O}(mn^3)$. However, in practice the algorithm runs much faster; we suspect a tighter bound exists which is not dependent on m.

Proposition 1. In cases where a classical difference constraint solver soundly proves unsatisfiability, wrapdiff_fixpoint also proves unsatisfiability.

Proof. Classical difference constraints are unsatisfiable if there is some cycle $C = [c_1, c_2, \ldots, c_k]$ in the graph, such that $S_C = \sum C > 0$. This conclusion is only sound if there is a corresponding cycle $C' = [-c'_k, \ldots, -c'_2, -c'_1]$ which prevents the cycle from wrapping to 0. Let $S_{C'} = \sum C'$, and let $S_C = pm + r$ such that $r \in [1, m-1]$. The cycle C excludes 0 only if $S_{C'} < (p+1)m$. Note that for each edge $c_i \in C$, $c_i - c'_i \leqslant 0$ (otherwise, the cycle $[c_i, c'_i]$ is trivially unsatisfiable).

Consider the behaviour of wrapdiff_fixpoint on the corresponding constraints $\{[c_1, c'_1]_m, [c_2, c'_2]_m, \ldots, [c_k, c'_k]_m\}$. The interval size $c'_i - c_i$ is non-negative. As $S_{C'} - S_C < m$, the size of each partial-sum interval $|\sum_{i=1}^{j} [c_i, c'_i]_m|$ is less than m, so the interval cannot wrap. Adding all the edges then yields the interval $[S_C, S_{C'}]_m$, which does not contain 0. If $0 \notin \sum_{i=1}^{k} [c_i, c'_i]_m$, we also have $\sum_{i=1}^{k-1} [c_i, c'_i]_m \sqcap [c_k, c'_k]_m = \bot$. ($A \sqcap -B \neq \bot$ means there is some x such that $x \in A, -x \in B$. Therefore $0 = x + (-x) \in A + B$.)

Hence, if a cycle exists which allows classical difference logic to soundly conclude unsatisfiability, wrapdiff_fixpoint will do the same. □

4 Experimental evaluation

In this section, we evaluate the performance of the two **SMT**-based methods, and the incomplete shortest-path approach. For the **SMT**($\mathcal{BV}$) approach, we used the STP solver [6]; for the **SMT**($\mathcal{DL}$) encoding, we used the Z3 theorem prover [3]. The shortest-path algorithm is implemented in C++. The evaluation was conducted on a 3.00GHz Core2 Duo with 2Gb of RAM running Ubuntu GNU/Linux 10.04. Reported times are in milliseconds.

We compared the performance of the two approaches on a set of randomly generated problems over $\mathbb{Z}_{2^{32}}$ with an increasing number of variables. 100 instances were generated for each problem size between 20 and 200 variables. To ensure a mix of satisfiable and unsatisfiable instances, the number of constraints $|C|$ was fixed to $1.2|V|$. Of these, $\frac{1}{10}$ are ordering constraints, the remainder being uniformly distributed proximity constraints.[4] Results are given in Table 1.

[4] The solver and instances are available at `ww2.cs.mu.oz.au/~ggange/moddiff/`

$\|V\|$	$\|C\|$	$\text{TIME}_{\mathcal{BV}}$	$\text{TIME}_{\mathcal{DL}}$	TIME_{fix}	#U	#FP
20	24	50.8	19.2	0.2	24	1
40	48	99.9	24.4	0.4	22	1
60	72	150.0	29.8	0.8	22	1
80	96	197.5	36.4	1.1	29	1
100	120	268.9	43.3	1.7	22	0
120	144	341.3	50.9	2.0	21	0
140	168	404.0	59.0	2.6	22	1
160	192	494.9	65.9	2.8	27	0
180	216	537.7	73.2	3.4	31	1
200	240	675.6	85.5	3.9	25	0

Table 1: Comparing the **SMT**($\mathcal{BV}$) and **SMT**($\mathcal{DL}$) approaches with wrapdiff_fixpoint. Time reported (in milliseconds) is the average runtime over 100 instances of each size.

$\text{TIME}_{\mathcal{BV}}$, $\text{TIME}_{\mathcal{DL}}$ and TIME_{fix} denote the time for each method to solve all instances of the given size. #U indicates the number of unsatisfiable instances, and #FP the number of instances which the fixpoint-based method incorrectly reported to be satisfiable.

On these instances, the **SMT**($\mathcal{DL}$) encoding is considerably faster than the **SMT**($\mathcal{BV}$) encoding. The incomplete method is generally around 30 times faster than the **SMT**($\mathcal{DL}$) method, while having a very low false positive rate.

5 Conclusion

Difference logic is useful for program verification and analysis. However, for machine-arithmetic-aware program analysis and verification, classical difference logic is unsound. We have shown that, when extended to modular arithmetic, difference constraints are NP-complete even for $\mathbb{Z}_3$. We have presented two complete methods based on **SMT** techniques, and a sound heuristic based on a fixpoint computation. The heuristic runs substantially faster than the complete methods, and correctly determines unsatisfiability for the majority of the random instances we tested. It would be interesting to develop alternative techniques which improve precision without sacrificing performance. Further work will involve embedding this method in an abstract interpretation framework for static analysis.

Acknowledgments

This work was supported through ARC grant DP110102579. We are grateful to the anonymous reviewers who identified a number of critical misprints in the draft version and suggested an improved SMT encoding which we have adopted.

References

1. N. Bjørner, A. Blass, Y. Gurevich, and M. Musuvathi. Modular difference logic is hard, November 2008. Unpublished, arXiv:0811.0987v1.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.
3. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
4. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 179–212. Springer, 1989.
5. R. W. Floyd. Algorithm 97: Shortest path. *Comm. ACM*, 5:345, 1962.
6. V. Ganesh and D. L. Dill. A decision procedure for bit-vectors and arrays. In *Computer Aided Verification*, volume 4590 of *LNCS*, pages 519–531. Springer, 2007.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.
8. A. Gotlieb, M. Leconte, and B. Marre. Constraint solving on modular integers. In *Proc. Ninth Int. Workshop Constraint Modelling and Reformulation*, 2010. `http://www.it.uu.se/research/group/astra/ModRef10/programme.html`.
9. A. K. John and S. Chakraborty. A quantifier elimination algorithm for linear modular equations and disequations. In *Computer Aided Verification*, volume 6806 of *LNCS*, pages 486–503. Springer, 2011.
10. D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. Springer, 2008.
11. A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
12. M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Automata, Languages and Combinatorics*, 7(3):321–350, 2002.
13. M. Müller-Olm and H. Seidl. Analysis of modular arithmetic. *ACM Trans. Programming Languages and Systems*, 29(5), 2007. Article 29.
14. J. A. Navas, P. Schachte, H. Søndergaard, and P. J. Stuckey. Signedness-agnostic program analysis: Precise integer bounds for low-level code. In *Proc. APLAS 2012*, volume 7705 of *LNCS*, pages 115–130. Springer, 2012.
15. J. Regehr and U. Duongsaa. Deriving abstract transfer functions for analyzing embedded software. In *LCTES'06: Proc. Conf. Language, Compilers, and Tool Support for Embedded Systems*, pages 34–43. ACM Press, 2006.
16. R. Sen and Y. N. Srikant. Executable analysis using abstract interpretation with circular linear progressions. In *Proc. Fifth IEEE/ACM Int. Conf. Formal Methods and Models for Codesign*, pages 39–48. IEEE, 2007.
17. A. Simon and A. King. Taming the wrapping of integer arithmetic. In *Static Analysis*, volume 4634 of *LNCS*, pages 121–136. Springer, 2007.
18. O. Strichman, S. A. Seshia, and R. E. Bryant. Deciding separation formulas with SAT. In *Computer Aided Verification*, volume 2404 of *LNCS*, pages 209–222. Springer, 2002.
19. C. Wang, F. Ivančić, and M. Ganai. Deciding separation logic formulae by SAT and incremental negative cycle elimination. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3835 of *LNAI*, pages 322–336. Springer, 2005.