TR-IIS-06-001

On the Satisfiability of Modular Arithmetic Formula

Bow-Yaw Wang

January 24, 2006 || Technical Report No. TR-IIS-06-001
http://www.iis.sinica.edu.tw/LIB/TechReport/tr2006/tr06.html

On the Satisfiability of Modular Arithmetic Formula

Bow-Yaw Wang

Institute of Information Science
Academia Sinica
Taiwan

January 24, 2006

Abstract. Modular arithmetic is the underlying integer computation model in conventional programming languages. In this paper, we discuss the satisfiability problem of modular arithmetic formulae over the finite ring $\mathbb{Z}_{2^\omega}$. Although an upper bound of $2^{2^{O(n^4)}}$ can be obtained by solving alternation-free Presburger arithmetic, it is easy to see that the problem is in fact **NP**-complete. Further, we give an efficient reduction to integer programming with the number of constraints and variables linear in the length of the given linear modular arithmetic formula. For non-linear modular arithmetic formulae, an additional factor of ω is needed. With the advent of efficient integer programming packages, our word-level encoding could be useful to software verification in practice.

1 Introduction

Modular arithmetic is widely used in the design of cryptosystems and pseudo random number generators. In the RSA public key system, exponentiation in modular arithmetic is needed in its encryption and decryption schemes [30]. And many pseudo random number generators use the linear congruent method [19]. Since integers are represented in finite bits in conventional programming languages such as C, modular arithmetic is often required in software verification as well. Consider the following C program.

```
void main (void) {
    int i = 0x87654321, j = 0xfedcba09;
    printf ("%u * %u = %u\n", i, j, i * j);
    printf ("%d * %d = %d\n", i, j, i * j);
}
```

Its output may look as follows.[1]

```
2271560481 * 4275878409 = 3283179049
-2023406815 * -19088887 = -1011788247
```

But $2271560481 \times 4275878409 = 9712916415445554729$ and $-2023406815 \times -19088887 = 38624584046564905$. On the other hand, it is straightforward to check 38624584046564905

[1] The program is compiled by gcc 4.0.2 and executed on a Fedora Core 4 Linux system.

$\equiv 9712916415445554729 \equiv 3283179049 \equiv -1011788247 \pmod{2^{32}}$. Signed and unsigned integral computations in C are modular arithmetic. Indeed, many algorithms are designed to avoid overflow in modular arithmetic explicitly. Verification tools therefore need to support modular arithmetic to check these algorithms.

However, modular arithmetic has yet to be found in modern software model checkers ([15, 3, 14] to name a few). Since it does not seem to be easy to perform modular arithmetic by binary encoding, recent development in BDD and SAT technologies does little help in this regard. Particularly, the abstraction of programs with modular arithmetic may still need user intervention. One therefore wonders whether efficient decision procedures can be deployed in the construction of the abstract program.

In this paper, we consider the satisfiability problem of propositional formulae with modular arithmetic. All arithmetic computation in the formulae is over the finite ring $\mathbb{Z}_{2^\omega}$ for some fixed ω. In addition to linear terms, non-linear terms such as term multiplications and modulo operations of terms are allowed. We show that the satisfiability problem is **NP**-complete for formulae of linear modular arithmetic. The problem is still in **NP** for non-linear expressions.

We give a reduction to integer programming to construct the decision procedure in practice. From theoretical point of view, a reduction to SAT is also possible. But it is only natural to exploit linear arithmetic in integer programming, rather than perform arithmetic computation bitwisely. Additionally, the problem of solving integral linear constraints has been investigated by the operation research community for decades. Many heuristics for integer programming are known, especially those by relaxation and rounding [11, 24]. Our reduction could exploit these heuristics unavailable to Boolean encoding.

In order to have practical decision procedures for modular arithmetic, several issues have to be addressed in our reduction. Firstly, modular computation must be simulated by linear constraints, as well as all logical operations. Furthermore, non-linear multiplications and modulo operations need be expressed in the form of linear constraints. Most importantly, we would not like our reduction to increase the size of the problem significantly. Our construction should not use more than linear number of constraints and variables in the length of the modular linear arithmetic formula.

It is well-known that the first-order non-linear arithmetic theory is undecidable [12]. Presburger arithmetic is a first-order linear arithmetic theory which is decidable [9, 22, 27]. In [22], Oppen shows an upper bound of $2^{2^{2^{cn \lg n}}}$ for determining the truth of Presburger arithmetic formula of length n. If the number of quantifier alternation is m, the problem can be solved in time $2^{2^{O(n^{m+4})}}$ and space $2^{O(n^{m+4})}$ [27]. Although Presburger arithmetic can express first-order linear arithmetic properties, it does not support modular arithmetic directly. The modular arithmetic found in conventional programming languages has to be encoded in the first-order linear arithmetic theory. But the doubly exponential time complexity makes this approach impractical.

Other approaches to solving Presburger arithmetic are available. In [5], a survey of the automata-theoretic approach is given. For a special class of quantifier-free Presburger arithmetic, [29] shows a reduction to Boolean satisfiability. Similar to [9, 22, 27], neither of them considers modular arithmetic nor non-linear expressions, however.

Integer programming models the optimization problem with a set of linear constraints [24, 11, 23]. The problem is known to be **NP**-complete. But it does not support modular arithmetic natively. It also does not consider arbitrary combinations of constraints, only their conjunction is allowed. [6] transforms conjunctions of linear constraints to integer programming.

In [13, 21, 20], the inference of the affine relations among integer variables in conventional languages is discussed. But inequalities and non-linear modular arithmetic are not considered. In [2], a decision procedure for modular arithmetic is proposed. Although the authors use a mathematical approach, the complexity of their decision procedure is not discussed. Besides, it is unclear whether the logical and modulo operations can be added in their framework.

We note that our reduction may also serve as a reduction to Presburger arithmetic. Since Presburger arithmetic does not allow modular arithmetic, encoding it in linear constraints allows us to solve the problem by various decision procedures for Presburger arithmetic. However, solving the corresponding Presburger arithmetic formula requires $2^{2^{O(n^4)}}$ in the length of the modular arithmetic formula. Our reduction is more efficient than this approach asymptotically.

The remaining of paper is organized as follows. Section 2 contains the background. It is followed by the syntax and semantics of linear modular arithmetic in Section 3. The algorithm for the satisfiability of linear modular arithmetic is presented in Section 4. Section 5 discusses the satisfiability problem for non-linear modular arithmetic. Applications of our algorithm are shown in 6. Finally, Section 7 concludes the paper.

2 Preliminaries

Let $\mathbb{Z}$ be the set of integers, $\mathbb{Z}^+$ the set of positive integers, and $\mathbb{Z}^\times$ the set of non-zero integers. In the following exposition, we will fix the set X of integer variables and $m = 2^\omega$ where $\omega \in \mathbb{Z}^+$.

Definition 1. *([17], for example) For any* $a \in \mathbb{Z}, b \in \mathbb{Z}^\times$*, there are* $q, r \in \mathbb{Z}$ *such that* $a = bq + r$ *and* $0 \leq r < b$*.*

The numbers q and r are called *a quotient b* (a quo b) and *a modulo b* (a mod b) respectively. For convenience, we define *signed quotient* and *signed modulo* as follows.

$$a \text{ smod } b \triangleq \begin{cases} a \bmod b & \text{if } 0 \leq a \bmod b < \lfloor \frac{|b|}{2} \rfloor \\ a \bmod b - |b| & \text{if } \lfloor \frac{|b|}{2} \rfloor \leq a \bmod b < |b| \end{cases}$$

$$a \text{ squo } b \triangleq \frac{a - (a \text{ smod } b)}{b}$$

For example, -7 quo $-3 = 3$ and -7 mod $-3 = 2$, but -7 squo $-3 = 2$ and -7 smod $-3 = -1$ for $-7 = -3 \times 3 + 2$. We say a is *congruent to b modulo m*, $a \equiv b \pmod{m}$, if $(a - b) \bmod m = 0$. For any $a \in \mathbb{Z}$, the *residue class of a modulo m* is the set $[a] \triangleq \{x | x \equiv a \pmod{m}\}$. It is easy to verify that the *residue class system* $\mathbb{Z}_m = (\{[0], [1], \ldots, [m-1]\}, +, [0], \cdot, [1])$ is a commutative ring [17].

Since $\mathbb{Z}_m$ consists of residue classes of integers modulo m, several representations of the equivalence classes are possible. Particularly, we call $\{-\frac{m}{2}, \ldots, -1, 0, 1, \frac{m}{2} - 1\}$ the *signed representation* and $\{0, 1, \ldots, m-1\}$ the *unsigned representation.*

To emulate integral computation in conventional languages, we use the signed representation if not mentioned otherwise. It is straightforward to support both signed and unsigned representations within our framework, though. If $c \in \mathbb{Z}$, the notation $c \in \mathbb{Z}_m$ denotes that c is an element in the signed representation of $\mathbb{Z}_m$.

Let $c, a_j \in \mathbb{Z}$ and $x_j \in X$ for $0 \leq j < N$. A *linear constraint* is of following form

$$\sum_{j=0}^{N-1} a_j x_j \sim c$$

where $\sim \in \{\leq, <, =, >, \geq\}$. Given a set of M linear constraints $\sum_{j=0}^{N-1} a_{i,j} x_j \sim_i c_i$ for $0 \leq i < M$, and a linear objective function $\sum_{j=0}^{N-1} b_j x_j$, the *integer programming problem* is to find a *valuation* $\rho : X \to \mathbb{Z}$ such that ρ satisfies all linear constraints and attains the maximum value of the objective function. We denote an instance of integer programming problem as follows.

$$\begin{array}{rl} \text{maximize} & \sum_{j=0}^{N-1} b_j x_j \\ \text{subject to} & \begin{array}{rcl} \sum_{j=0}^{N-1} a_{0,j} x_j & \sim_0 & c_0 \\ \sum_{j=0}^{N-1} a_{1,j} x_j & \sim_1 & c_1 \\ & \vdots & \\ \sum_{j=0}^{N-1} a_{M-1,j} x_j & \sim_{M-1} & c_{M-1} \end{array} \end{array}$$

It is known that the integer programming problem is **NP**-complete [23]. If the range of the valuation is relaxed to be rational, it is called the *linear programming problem.* It is also known that the linear programming problem is in **P** [23]. There are heuristics for integer programming by relaxing the problem to linear programming [24, 11].

3 Linear Modular Arithmetic

$$\begin{aligned}
\text{Term } t &\triangleq c \mid c \cdot t \mid t \,\%\, c \mid t \div c \mid t + t' \\
\text{Atomic Proposition } l &\triangleq \mathsf{ff} \mid t \leq t' \mid t = t' \\
\text{Formula } f &\triangleq l \mid \neg f \mid f \wedge f' \mid f \vee f'
\end{aligned}$$

Fig. 1. Syntax of Linear Modular Arithmetic Formula over $\mathbb{Z}_m$

For any $c \in \mathbb{Z}_m$ and $x \in X$, define the syntax of a Linear Modular Arithmetic Formula over $\mathbb{Z}_m$ in Figure 1. We use the symbols $\%$ and $\div$ for the modulo and quotient operators respectively in our object language to avoid confusion. Also, we do not use syntactic translation for equality nor any of the logical connectives. A more efficient reduction can be attained by treating each operator separately, although it does not improve the performance asymptotically. Finally, only constants in $\mathbb{Z}_m$ are allowed. Overflowed constants would produce warnings by compilers.[2] They could be identified rather easily.

$$\begin{aligned}
[\![c]\!]_\rho &\triangleq c & [\![t \,\%\, c]\!]_\rho &\triangleq [\![t]\!]_\rho \bmod c \\
[\![c \cdot t]\!]_\rho &\triangleq c[\![t]\!]_\rho \text{ smod } m & [\![t \div c]\!]_\rho &\triangleq [\![t]\!]_\rho \text{ quo } c \\
[\![t + t']\!]_\rho &\triangleq [\![t]\!]_\rho + [\![t']\!]_\rho \text{ smod } m
\end{aligned}$$

$$\begin{aligned}
[\![\mathsf{ff}]\!]_\rho &\triangleq \mathsf{false} & [\![\neg f]\!]_\rho &\triangleq \neg[\![f]\!]_\rho \\
[\![t \leq t']\!]_\rho &\triangleq [\![t]\!]_\rho \leq [\![t']\!]_\rho & [\![f \wedge f']\!]_\rho &\triangleq [\![f]\!]_\rho \wedge [\![f']\!]_\rho \\
[\![t = t']\!]_\rho &\triangleq [\![t]\!]_\rho = [\![t']\!]_\rho & [\![f \vee f']\!]_\rho &\triangleq [\![f]\!]_\rho \vee [\![f']\!]_\rho
\end{aligned}$$

Fig. 2. Semantics of Linear Modular Arithmetic Formula over $\mathbb{Z}_m$

Let ρ be a valuation. The semantic function $[\![\bullet]\!]_\rho$ for linear modular arithmetic formulae over $\mathbb{Z}_m$ is defined in Figure 2. Since $c \in \mathbb{Z}_m$, it is unnecessary to compute the representative of the result for constants, modulo and quotient operations. For the other cases, their semantic values are obtained by signed modulo m.

Assume each integral and logical computation in a typical conventional language takes $O(1)$ time.[3] We can now phrase the *satisfiability problem* as follows.

Problem 1. (Satisfiability) Given a linear modular arithmetic formula f over $\mathbb{Z}_m$ with variables $\bar{x}$, determine whether there is a valuation ρ such that $[\![f]\!]_\rho = \mathsf{true}$.

[2] In gcc 4.0.2, the warning message "integer constant is too large for its type" is shown.

[3] For languages with infinite-precision integers such as Scheme [18], this assumption does not hold.

Since the evaluation of any linear modular arithmetic formula is in **P**, we immediately have the following upper bound for the satisfiability problem.

Proposition 1. *The satisfiability problem for any linear modular arithmetic formula f can be decided in **NP**.*

Proof. We first guess $|\bar{x}|$ numbers from $\mathbb{Z}_m$ in the valuation ρ, and then evaluate $[\![f]\!]_\rho$. □

The lower bound of the problem can be obtained by reduction from **3CNF**. Although Boolean variables are not allowed in linear modular arithmetic, they can be simulated by the parity of integer variables fairly easily.

Proposition 2. *The satisfiability problem for any linear modular arithmetic formula f is **NP**-hard.*

Proof. We reduce **3CNF** to the satisfiability problem as follows. For any Boolean variable b_i, we use the atomic proposition $x_i \,\%\, 2 = 1$ instead. For example, the clause $b_0 \vee \neg b_1 \vee b_2$ is translated to

$$(x_0 \,\%\, 2 = 1) \vee \neg(x_1 \,\%\, 2 = 1) \vee (x_2 \,\%\, 2 = 1).$$

If there is a satisfying truth assignment to b_i's, we choose

$$x_i = \begin{cases} 1 \text{ if } b_i = \mathsf{true} \\ 0 \text{ otherwise} \end{cases}$$

Conversely, if there is a satisfying valuation to x_i's, we choose

$$b_i = \begin{cases} \mathsf{true} \text{ if } x_i \text{ is odd} \\ \mathsf{false} \text{ otherwise} \end{cases}$$

□

Corollary 1. *The satisfiability problem for linear modular arithmetic formula is **NP**-complete.*

4 Solving the Satisfiability Problem for Linear Modular Arithmetic

Since modular arithmetic is the default integral computation in conventional languages, deciding the satisfiability of linear modular arithmetic formula could be useful in software verification. One may, of course, use binary encoding and solve the problem in the Boolean domain. But it would disregard the underlying mathematical nature of the problem. We are therefore looking for alternatives capable of exploiting the underlying mathematical structure of the problem.

Integer programming is chosen for the following reasons. Firstly, linear arithmetic is primitive to the problem. The underlying mathematical structure can be reflected in the reduction naturally. Secondly, heuristics for integer programming are known, especially those by relaxation and rounding [24]. Some instances of the integer programming problem may be solved by linear programming efficiently in practice. Lastly, optimized integer programming packages are available [25, 26]. They could be viable alternatives to the traditional formal verification techniques.

Given an instance of any syntactic class (terms, atomic propositions, or formulae), we translate it to an integer variable and a set of constraints. Intuitively, the integer variable has the semantic value of the given instance for any valuation subjecting to the set of constraints. For terms, the integer variable has a value in $[-\frac{m}{2}, \frac{m}{2} - 1]$. For atomic propositions and formulae, it has values 0 or 1.

$$\sigma(c) \triangleq (p, p = c) \qquad \sigma(c \cdot t) \triangleq \begin{pmatrix} & \alpha \\ p, & -\frac{m}{2} \leq p < \frac{m}{2} \\ & cp' - mq = p \end{pmatrix} \text{ where } (p', \alpha) = \sigma(t)$$

$$\sigma(t_0 + t_1) \triangleq \begin{pmatrix} & \alpha_0 \\ & \alpha_1 \\ p, & -\frac{m}{2} \leq p < \frac{m}{2} \\ & p_0 + p_1 - mq = p \end{pmatrix} \text{ where } \begin{array}{l} (p_0, \alpha_0) = \sigma(t_0) \\ (p_1, \alpha_1) = \sigma(t_1) \end{array} \qquad \sigma(t \,\%\, c) \triangleq \begin{pmatrix} & \alpha \\ p, & 0 \leq p < |c| \\ & p' - cq = p \end{pmatrix} \text{ where } (p', \alpha) = \sigma(t)$$

$$\sigma(t \div c) \triangleq \begin{pmatrix} & \alpha \\ p, & 0 \leq p' - cp < |c| \end{pmatrix} \text{ where } (p', \alpha) = \sigma(t)$$

Fig. 3. Linear Constraints for Terms

Consider, for example, the following translation of $t \,\%\, c$ (Figure 3).

$$\begin{pmatrix} & \alpha \\ p, & 0 \leq p < |c| \\ & p' - cq = p \end{pmatrix} \text{ where } (p', \alpha) = \sigma(t)$$

The semantic value p' and constraints α of t are obtained by $\sigma(t)$ recursively. Since the semantic value p of $t \,\%\, c$ is equal to $p' \,\%\, c$, we add the constraints $0 \leq p < |c|$ and $p' - cq = p$. The following lemma shows that the semantics of terms is still retained in spite of the constraints in Figure 3.

Lemma 1. *Let t be a term in linear modular arithmetic and $(p, \alpha) = \sigma(t)$. For any valuation ρ, there is a valuation η such that*

– *η satisfies α;*
– *$\eta(p) = [\![t]\!]_\rho$; and*
– *$\eta(x) = \rho(x)$ for any x appearing in t.*

Proof. We proceed by induction. The basis follows from the definition immediately. For the inductive step, consider the following four cases.

1. $t_0 + t_1$. Let $(p_0, \alpha_0) = \sigma(t_0)$ and $(p_1, \alpha_1) = \sigma(t_1)$. There are valuations η_0 and η_1 such that $\eta_0(p_0) = [\![t_0]\!]_\rho$ and $\eta_1(p_1) = [\![t_1]\!]_\rho$. Define η as follows.
   – $\eta(y) = \eta_0(y)$ for all y appearing in α_0;
   – $\eta(y) = \eta_1(y)$ for all y appearing in α_1;
   – $\eta(p) = \eta_0(p_0) + \eta_1(p_1)$ smod m; and
   – $\eta(q) = \eta_0(p_0) + \eta_1(p_1)$ squo m.

   Then η is the desired valuation.
2. $c \cdot t$. Similar to the previous case.
3. $t \,\%\, c$. There is a valuation η' such that $\eta'(p') = [\![t]\!]_\rho$ by inductive hypothesis. Define $\eta(y) = \eta'(y)$ for all y appearing in α, $\eta(p) = \eta'(p')$ mod c, and $\eta(q) = \eta'(p')$ quo c.
4. $t \div c$. Similar to the previous case. □

Since any valuation satisfying additional constraints is still a valuation, the following lemma shows the constraints in Figure 3 do not change the semantics of terms.

Lemma 2. *Let t be a term in linear modular arithmetic and $(p, \alpha) = \sigma(t)$. For any valuation ρ satisfying α, we have $[\![t]\!]_\rho = \rho(p)$.*

Proof. We proceed by induction. The verification is routine. □

Proposition 3. *Let t be a term in linear modular arithmetic. The following statements are equivalent.*

– *There is a valuation ρ such that $[\![t]\!]_\rho = d$.*
– *There is a valuation η such that η satisfies α and $\eta(p) = d$ where $(p, \alpha) = \sigma(t)$.*

Proof. By Lemma 1 and 2. □

For atomic propositions, observe that

$$-m < -m + 1 = -\frac{m}{2} - (\frac{m}{2} - 1) \leq [\![t_0]\!]_\rho - [\![t_1]\!]_\rho \leq (\frac{m}{2} - 1) - (-\frac{m}{2}) = m - 1 < m.$$

Consider the atomic proposition $t_0 \leq t_1$. From Figure 4, we have

$$\left(p, \begin{array}{r} \alpha_0 \\ \alpha_1 \\ 0 \leq p \leq 1 \\ p_0 - p_1 - (m-1)(1-p) \leq 0 \\ p_0 - p_1 + mp > 0 \end{array} \right) \text{ where } \begin{array}{l} (p_0, \alpha_0) = \sigma(t_0) \\ (p_1, \alpha_1) = \sigma(t_1) \end{array}.$$

$$\lambda(\mathsf{ff}) \triangleq (p, p = 0)$$

$$\lambda(t_0 \le t_1) \triangleq \left(p, \begin{array}{r} \alpha_0 \\ \alpha_1 \\ 0 \le p \le 1 \\ p_0 - p_1 - (m-1)(1-p) \le 0 \\ p_0 - p_1 + mp > 0 \end{array} \right)$$

$$\text{where } \begin{array}{l} (p_0, \alpha_0) = \sigma(t_0) \\ (p_1, \alpha_1) = \sigma(t_1) \end{array}$$

$$\lambda(t_0 = t_1) \triangleq \left(p, \begin{array}{r} \alpha_0 \\ \alpha_1 \\ 0 \le q_0 + q_1 \le 1 \\ p_0 - p_1 + m(1-q_0) - q_0 \ge 0 \\ p_0 - p_1 - m(1-q_1) + q_1 \le 0 \\ p_0 - p_1 - m(q_0+q_1) \le 0 \\ p_0 - p_1 + m(q_0+q_1) \ge 0 \\ 1 - q_0 - q_1 = p \end{array} \right)$$

$$\text{where } \begin{array}{l} (p_0, \alpha_0) = \sigma(t_0) \\ (p_1, \alpha_1) = \sigma(t_1) \end{array}$$

Fig. 4. Linear Constraints for Atomic Propositions

Since the variables p_0 and p_1 have the semantic values of the terms t_0 and t_1 respectively, we have $-m < p_0 - p_1 \le m - 1$. If $p_0 \le p_1$, it is easy to verify that the constraints are satisfied if we choose the semantic value p to be 1. Conversely, if the variable p has the value 1, $p_0 - p_1 - (m-1)(1-p) = p_0 - p_1 \le 0$ is enforced by the constraints. Thus $p_0 \le p_1$.

For equality, one could use a less efficient construction by conjunctions and comparisons. But we have a slightly better translation in Figure 4. Intuitively, the variables q_0 and q_1 denote $p_0 > p_1$ and $p_0 < p_1$ respectively. Hence at most one of q_0 and q_1 can be 1. And the semantic value of $t_0 = t_1$ is 1 if and only if $q_0 = q_1 = 0$, namely, $1 - q_0 - q_1$.

The following lemma shows that we can replace the semantics values of atomic propositions by 0 or 1.

Lemma 3. *Let l be an atomic proposition in linear modular arithmetic and $(p, \alpha) = \lambda(l)$. For any valuation ρ, there is a valuation η such that*

– *η satisfies α;*
– *$[\![l]\!]_\rho = \mathsf{false}$ implies $\eta(p) = 0$; and*
– *$[\![l]\!]_\rho = \mathsf{true}$ implies $\eta(p) = 1$.*

Proof. For the terms t_0 and t_1 in any atomic proposition, there are valuations η_0 and η_1 such that

– $\eta_0(p_0) = [\![t_0]\!]_\rho$, η_0 satisfies α_0; and
– $\eta_1(p_1) = [\![t_1]\!]_\rho$, η_1 satisfies α_1

by Proposition 3, where $(p_0, \alpha_0) = \sigma(t_0)$ and $(p_1, \alpha_1) = \sigma(t_1)$. Define $\eta(y) = \eta_0(y)$ for any variable y appearing in α_0, and $\eta(y) = \eta_1(y)$ for any variable y appearing in α_1.

Consider the following cases.

1. ff. Trivial.
2. $t_0 \leq t_1$. If $[\![t_0]\!]_\rho \leq [\![t_1]\!]_\rho$, define $\eta(p) = 1$. Then the constraints $p_0 - p_1 - (m-1)(1-p) \leq 0$ and $p_0 - p_1 + mp > 0$ reduce to $\eta(p_0) - \eta(p_1) \leq 0$ and $\eta(p_0) - \eta(p_1) + m > 0$. Since $\eta(p_0) = [\![t_0]\!]_\rho \geq -\frac{m}{2}$ and $\eta(p_1) = [\![t_1]\!]_\rho < \frac{m}{2}$, we have $\eta(p_0) - \eta(p_1) > -m$.
   If $[\![t_0]\!]_\rho > [\![t_1]\!]_\rho$, define $\eta(p) = 0$. Then the constraints reduce to $\eta(p_0) - \eta(p_1) - (m-1) \leq 0$ and $\eta(p_0) - \eta(p_1) > 0$. Since $\eta(p_0) < \frac{m}{2}$ and $\eta(p_1) \geq -\frac{m}{2}$, $\eta(p_0) - \eta(p_1) < m$. Thus $\eta(p_0) - \eta(p_1) \leq m - 1$.
3. $t_0 = t_1$. Define

$$\eta(q_0) = \begin{cases} 1 & \text{if } \rho(p_0) > \rho(p_1) \\ 0 & \text{otherwise} \end{cases}$$
$$\eta(q_1) = \begin{cases} 1 & \text{if } \rho(p_1) > \rho(p_0) \\ 0 & \text{otherwise} \end{cases}$$
$$\eta(p) = 1 - \eta(q_0) - \eta(q_1)$$

   If $\rho(p_0) > \rho(p_1)$, then the constraints

$$\begin{aligned} p_0 - p_1 + m(1 - q_0) - q_0 &\geq 0 \\ p_0 - p_1 - m(1 - q_1) - q_1 &\leq 0 \\ p_0 - p_1 - m(q_0 + q_1) &\leq 0 \\ p_0 - p_1 + m(q_0 + q_1) &\geq 0 \end{aligned}$$

   reduce to

$$\begin{aligned} \eta(p_0) - \eta(p_1) - 1 &\geq 0 \\ \eta(p_0) - \eta(p_1) - m &\leq 0 \\ \eta(p_0) - \eta(p_1) + m &\geq 0. \end{aligned}$$

   Note that $-m \leq \eta(p_0) - \eta(p_1) \leq m$ for $-\frac{m}{2} \leq \eta(p_0) = [\![t_0]\!]_\rho, \eta(p_1) = [\![t_1]\!]_\rho < \frac{m}{2}$. Since $[\![t_0]\!]_\rho > [\![t_1]\!]_\rho$, $\eta(p_0) - \eta(p_1) \geq 1$ as well.
   The case for $\rho(p_1) > \rho(p_0)$ can be proved similarly.

When $[\![t_0]\!]_\rho = [\![t_1]\!]_\rho$, we have $\eta(q_0) = \eta(q_1) = 0$. The constraints reduce to

$$\begin{aligned}
\eta(p_0) - \eta(p_1) + m &\geq 0 \\
\eta(p_0) - \eta(p_1) - m &\leq 0 \\
\eta(p_0) - \eta(p_1) &\leq 0 \\
\eta(p_0) - \eta(p_1) &\geq 0.
\end{aligned}$$

And the result follows. □

Conversely, we can show that the constraints in Figure 4 reflect the semantic values of atomic propositions.

Lemma 4. *Let l be an atomic proposition in linear modular arithmetic and $(p, \alpha) = \lambda(l)$. For any valuation ρ satisfying α, we have*

- $\rho(p) = 0$ *implies* $[\![l]\!]_\rho =$ *false; and*
- $\rho(p) = 1$ *implies* $[\![l]\!]_\rho =$ *true.*

Proof. Note that $\rho(p_0) = [\![t_0]\!]_\rho$ and $\rho(p_1) = [\![t_1]\!]_\rho$ by Proposition 3.

1. $t_0 \leq t_1$. If $\rho(p) = 0$, the constraint $p_0 - p_1 + mp > 0$ reduces to $\rho(p_0) - \rho(p_1) > 0$. Hence $\rho(p_0) = [\![t_0]\!]_\rho > \rho(p_1) = [\![t_1]\!]_\rho$.
   If $\rho(p) = 1$, the constraint $p_0 - p_1 - (m-1)(1-p) \leq 0$ reduces to $\rho(p_0) - \rho(p_1) \leq 0$. Hence $\rho(p_0) = [\![t_0]\!]_\rho \leq \rho(p_1) = [\![t_1]\!]_\rho$.
2. $t_0 = t_1$. Note that $0 \leq \rho(p) \leq 1$ for $0 \leq \rho(q_0) + \rho(q_1) \leq 1$.
   If $\rho(p) = 1$, then $\rho(q_0) = \rho(q_1) = 0$. Hence the constraints

$$\begin{aligned}
p_0 - p_1 - m(q_0 + q_1) &\leq 0 \\
p_0 - p_1 + m(q_0 + q_1) &\geq 0
\end{aligned}$$

   reduce to $\rho(p_0) - \rho(p_1) = 0$. Therefore $\rho(p_0) = [\![t_0]\!]_\rho = \rho(p_1) = [\![t_1]\!]_\rho$ as required.
   On the other hand, suppose $\rho(q_0) = 1$ but $\rho(q_1) = 0$. Then the constraint $p_0 - p_1 + m(1 - q_0) - q_0 \geq 0$ reduces to $\rho(p_0) - \rho(p_1) \geq 1$. Hence $\rho(p_0) = [\![t_0]\!]_\rho > \rho(p_1) = [\![t_1]\!]_\rho$. The other case is similar. □

Proposition 4. *Let l be an atomic proposition in linear modular arithmetic. Then*

1. *there is a valuation ρ such that $[\![l]\!]_\rho =$ true $\Leftrightarrow$ there is a valuation η such that η satisfies α and $\eta(p) = 1$ where $(p, \alpha) = \lambda(l)$.*

$$\phi(l) \triangleq \lambda(l) \qquad \phi(\neg f) \triangleq \begin{pmatrix} p, & \begin{matrix} \alpha \\ 1 - p' = p \end{matrix} \end{pmatrix} \text{ where } (p', \alpha) = \phi(f)$$

$$\phi(f_0 \wedge f_1) \triangleq \begin{pmatrix} & \alpha_0 \\ & \alpha_1 \\ p, & 0 \leq p \leq 1 \\ & p_0 + p_1 \geq 2p \\ & p_0 + p_1 \leq 1 + p \end{pmatrix} \text{ where } \begin{matrix} (p_0, \alpha_0) = \phi(f_0) \\ (p_1, \alpha_1) = \phi(f_1) \end{matrix} \qquad \phi(f_0 \vee f_1) \triangleq \begin{pmatrix} & \alpha_0 \\ & \alpha_1 \\ p, & 0 \leq p \leq 1 \\ & p_0 + p_1 \geq p \\ & p_0 + p_1 \leq 2p \end{pmatrix} \text{ where } \begin{matrix} (p_0, \alpha_0) = \phi(f_0) \\ (p_1, \alpha_1) = \phi(f_1) \end{matrix}$$

Fig. 5. Linear Constraints for Formulae

2. *there is a valuation ρ such that $[\![l]\!]_\rho = \textsf{false} \Leftrightarrow$ there is a valuation η such that η satisfies α and $\eta(p) = 0$ where $(p, \alpha) = \lambda(l)$.*

Proof. By Lemma 3 and 4. □

Let p_0 and p_1 be the semantic values of the subformulae f_0 and f_1 respectively. Consider the constraints in the translation of their disjunction (Figure 5).

$$p_0 + p_1 \geq p$$
$$p_0 + p_1 \leq 2p$$

We would like the semantic value p of their disjunction to be 0 when both p_0 and p_1 are 0. The constraint $p_0 + p_1 \geq p$ forces p to be 0 when p_0 and p_1 are 0. On the other hand, p may be 0 or 1 when any of the disjuncts is true. We therefore add a lower bound of p by the constraint $p_0 + p_1 \leq 2p$.

Note that we do not translate the input formula to canonical forms. Since the translation could increase the length of the formula significantly, it would not be efficient. In order to have linear number of constraints and variables, it is crucial not to rearrange the input formula to canonical forms.

Lemma 5. *Let f be a formula in linear modular arithmetic and $(p, \alpha) = \phi(f)$. For any valuation ρ, there is a valuation η such that*

- *η satisfies α;*
- *$[\![f]\!]_\rho = \textsf{false}$ implies $\eta(p) = 0$; and*
- *$[\![f]\!]_\rho = \textsf{true}$ implies $\eta(p) = 1$.*

Proof. For atomic propositions and negations, the lemma follows from Proposition 4 and inductive hypothesis immediately.

For conjunctions and disjunctions, there are valuations η_0 and η_1 such that

- η_0 satisfies α_0, $[\![f_0]\!]_\rho = \mathsf{false}$ implies $\eta_0(p_0) = 0$, $[\![f_0]\!]_\rho = \mathsf{true}$ implies $\eta_0(p_0) = 1$; and
- η_1 satisfies α_1, $[\![f_1]\!]_\rho = \mathsf{false}$ implies $\eta_1(p_1) = 0$, $[\![f_1]\!]_\rho = \mathsf{true}$ implies $\eta_1(p_1) = 1$.

Define $\eta(y) = \eta_0(y)$ for any variable y occurring in α_0 and $\eta(y) = \eta_1(y)$ for any variable y occurring in α_1.

1. $f_0 \wedge f_1$. If one of $[\![f_0]\!]_\rho$ and $[\![f_1]\!]_\rho$ is false, define $\eta(p) = 0$. We have $0 \leq \eta(p_0) + \eta(p_1) \leq 1$. Hence $2\eta(p) \leq \eta(p_0) + \eta(p_1) \leq 1 + \eta(p)$.
   If $[\![f_0]\!]_\rho = [\![f_1]\!]_\rho = \mathsf{true}$, define $\eta(p) = 1$. We have $\eta(p_0) + \eta(p_1) = 2$. Hence $2\eta(p) \leq \eta(p_0) + \eta(p_1) \leq 1 + \eta(p)$.
   In both cases, the following constraints are satisfied.
$$p_0 + p_1 \geq 2p$$
$$p_0 + p_1 \leq 1 + p$$
2. $f_0 \vee f_1$. If one of $[\![f_0]\!]_\rho$ and $[\![f_1]\!]_\rho$ is true, define $\eta(p) = 1$. We have $1 \leq \eta(p_0) + \eta(p_1) \leq 2$. Hence $\eta(p) \leq \eta(p_0) + \eta(p_1) \leq 2\eta(p)$.
   If $[\![f_0]\!]_\rho = [\![f_1]\!]_\rho = \mathsf{false}$, define $\eta(p) = 0$. We have $\eta(p_0) + \eta(p_1) = 0$. Hence $\eta(p) \leq \eta(p_0) + \eta(p_1) \leq 2\eta(p)$.
   Hence, the following constraints are satisfied as required.
$$p_0 + p_1 \geq p$$
$$p_0 + p_1 \leq 2p$$

□

All logical operations in linear modular arithmetic are in fact mimicked by the constraints in Figure 4.

Lemma 6. *Let f be a formula in linear modular arithmetic and $(p, \alpha) = \phi(f)$. For any valuation ρ satisfying α, we have*

- *$\rho(p) = 0$ implies $[\![f]\!]_\rho = \mathsf{false}$; and*
- *$\rho(p) = 1$ implies $[\![f]\!]_\rho = \mathsf{true}$.*

Proof. The lemma follows from Proposition 4 and inductive hypothesis respectively for atomic propositions and negations.

For $f_0 \wedge f_1$ and $f_0 \vee f_1$, we have

– $\rho(p_0) = 0$ implies $[\![f_0]\!]_\rho = \mathsf{false}$, $\rho(p_0) = 1$ implies $[\![f_0]\!]_\rho = \mathsf{true}$; and
– $\rho(p_1) = 0$ implies $[\![f_1]\!]_\rho = \mathsf{false}$, $\rho(p_1) = 1$ implies $[\![f_1]\!]_\rho = \mathsf{true}$

by inductive hypothesis.

1. $f_0 \wedge f_1$. Suppose $\rho(p) = 0$. Then $0 \le \rho(p_0) + \rho(p_1) \le 1$ by the following constraints
$$p_0 + p_1 \ge 2p$$
$$p_0 + p_1 \le 1 + p.$$
Hence $\rho(p_0) = 0$ or $\rho(p_1) = 0$. Thus $[\![f_0]\!]_\rho = \mathsf{false}$ or $[\![f_1]\!]_\rho = \mathsf{false}$. $[\![f_0 \wedge f_1]\!]_\rho = \mathsf{false}$ as required.
On the other hand, $\rho(p) = 1$ implies $2 \le \rho(p_0) + \rho(p_1) \le 2$. Hence $[\![f_0]\!]_\rho = [\![f_1]\!]_\rho = \mathsf{true}$.
2. $f_0 \vee f_1$. Suppose $\rho(p) = 0$. Then $0 \le \rho(p_0) + \rho(p_1) \le 0$ by
$$p_0 + p_1 \ge p$$
$$p_0 + p_1 \le 2p.$$
Therefore $[\![f_0]\!]_\rho = [\![f_1]\!]_\rho = \mathsf{false}$.
Finally, $\rho(p) = 1$ implies $1 \le \rho(p_0) + \rho(p_1) \le 2$. Hence $[\![f_0]\!]_\rho = \mathsf{true}$ or $[\![f_1]\!]_\rho = \mathsf{true}$ as required. □

Given a formula in linear modular arithmetic, there is a set of constraints such that the semantic value of the formula is denoted by the designated variable in our construction. Our progress is summarized in the following proposition.

Proposition 5. *Let f be a formula in linear modular arithmetic. Then*

1. *there is a valuation ρ such that $[\![f]\!]_\rho = \mathsf{true} \Leftrightarrow$ there is a valuation η such that η satisfies α and $\eta(p) = 1$ where $(p, \alpha) = \phi(f)$.*
2. *there is a valuation ρ such that $[\![f]\!]_\rho = \mathsf{false} \Leftrightarrow$ there is a valuation η such that η satisfies α and $\eta(p) = 0$ where $(p, \alpha) = \phi(f)$.*

Proof. By Lemma 5 and 6 □

To decide the satisfiability problem of a linear modular arithmetic formula f, we first obtain an integer variable p and a set of constraints α from the translation $\phi(f)$. It is straightforward to see that the satisfiability problem can be solved by optimizing the objective function p with respect to α.

Our translation is constructed recursively. A recursive call is invoked for each subformula in the input formula. Further, a constant number of constraints and variables are added in each recursion. Since the number of subformulae is linear in the length of the input formula, the corresponding integer programming problem has the number of variables and constraints linear in the length of the input formula. The following theorem summarizes our result on the satisfiability problem for linear modular arithmetic formulae.

Theorem 1. *Given a formula f in linear modular arithmetic, the satisfiability problem can be solved by an instance of the integer programming problem with the number of constraints and variables linear in $|f|$.*

Proof. Let $(p, \alpha) = \phi(f)$. Consider the following integer programming problem.

$$\begin{gathered}\text{maximize } p \\ \text{subject to } \alpha\end{gathered}$$

If f is satisfiable, there is a valuation η satisfying α with $\eta(p) = 1$ by Proposition 5. On the other hand, if there is a satisfying valuation η with $\eta(p) = 1$, then f is satisfiable by Proposition 5 as well.

In the construction of α, at most 3 variables and 7 constraints are added for each subterms or subformulae in f. Since the number of subterm or subformula in f is linear in $|f|$, the number of variables and constraints in α is linear in $|f|$. $\square$

5 Solving the Satisfiability Problem for Modular Arithmetic

Based on the translation of linear modular arithmetic formulae, multiplications and modulo operations of arbitrary terms can be emulated in integer programming. Of course, one could use the binary representation and encode a multiplication circuit in linear modular arithmetic. But it would introduce many temporary variables. Besides, the mathematical nature of the problem would not be preserved by Boolean encoding. We hereby propose a more efficient translation.

$$\text{Term } t \triangleq \ldots \mid t \cdot t' \mid t \,\%\, t'$$

$$\begin{aligned}\llbracket t \cdot t' \rrbracket_\rho &\triangleq \llbracket t \rrbracket_\rho \llbracket t' \rrbracket_\rho \text{ smod } m \\ \llbracket t \,\%\, t' \rrbracket_\rho &\triangleq \llbracket t \rrbracket_\rho \text{ mod } \llbracket t' \rrbracket_\rho\end{aligned}$$

Fig. 6. Syntax and Semantics of Modular Arithmetic over $\mathbb{Z}_m$

The syntax and semantics of modular arithmetic extend those of linear modular arithmetic by term multiplication, $t \cdot t'$, and modulo operation of terms, $t \bmod t'$, (Figure 6).

Similar to linear terms, the semantic value of term multiplication uses signed modulo to reflect the semantics of conventional programming languages (as illustrated in Introduction). On the other hand, it is unnecessary to compute the signed representation for modulo operations of terms since overflow would not occur.

Theorem 2. *The satisfiability problem for modular arithmetic formula is* ***NP****-complete.*

Proof. The lower bound follows from Proposition 2. The upper bound is also obtained by nondeterministic evaluation. □

In order to compute non-linear terms, we will find the binary representations of operands' semantic values first. But it becomes complicated for negative numbers. However, it is safe to use unsigned representation in this context. Observe that

$$ab \equiv (a+m)b \equiv a(b+m) \equiv (a+m)(b+m) \pmod{m}.$$

We therefore assume the unsigned representation, compute the result, then convert it back to the signed representation for multiplications of terms. Thus, only the linear constraints for unsigned multiplication is needed.

$$\chi(p_0, p_1) \triangleq \left(c, \begin{array}{r@{\;}l} p_1 &< m \\ 0 \le b_i &\le 1 \text{ for } 0 \le i < \omega \\ \sum_{i=0}^{\omega-1} 2^i b_i &= p_0 \\ 0 \le c_i &\le 2^i p_1 \text{ for } 0 \le i < \omega \\ 2^i p_1 - 2^i m(1-b_i) &\le c_i \text{ for } 0 \le i < \omega \\ 2^i m b_i &\ge c_i \text{ for } 0 \le i < \omega \\ \sum_{i=0}^{\omega-1} c_i &= c \end{array} \right)$$

Fig. 7. Linear Constraints for Unsigned Multiplication

More concretely, suppose $0 \le p_0 < m$. The following constraints compute the unsigned representation of p_0 (Figure 7).

$$\begin{array}{r@{\;}l} 0 \le b_i &\le 1 \text{ for } 0 \le i < \omega \\ \sum_{i=0}^{\omega-1} 2^i b_i &= p_0 \end{array}$$

Intuitively, the bit string $b_{\omega-1}b_{\omega-2}\cdots b_1 b_0$ is the binary representation for p_0.

To compute the partial result $c_i = 2^i b_i p_1$, we use the following constraints.

$$\begin{array}{r@{\;}l} 0 \le c_i &\le 2^i p_1 \\ 2^i p_1 - 2^i m(1-b_i) &\le c_i \\ 2^i m b_i &\ge c_i. \end{array}$$

If $b_i = 0$, we have $2^i m b_i = 0 \geq c_i \geq 0$. On the other hand, we have $2^i p_1 - 2^i m(1 - b_i) = 2^i p_1 \leq c_i \leq 2^i p_1$ when $b_i = 1$. Thus, $c_i = 2^i b_i p_1$.

Lemma 7. *Let p_0, p_1 be variables and $(p, \alpha) = \chi(p_0, p_1)$. For any valuation ρ where $0 \leq \rho(p_0), \rho(p_1) < m$, there is a valuation η such that*

- *$\eta(p_0) = \rho(p_0)$, $\eta(p_1) = \rho(p_1)$;*
- *η satisfies α; and*
- *$\eta(p) = \rho(p_0)\rho(p_1)$.*

Proof. Define η as follows.

- $\eta(p_0) = \rho(p_0)$;
- $\eta(p_1) = \rho(p_1)$;
- $\eta(b_i) = (\eta_0(p_0) \bmod 2^{i+1})$ quo 2^i for $0 \leq i < \omega$;
- For $0 \leq i < \omega$, $\eta(c_i) = 0$ if $\eta(b_i) = 0$; $\eta(c_i) = 2^i \eta(p_1)$ if $\eta(b_i) = 1$;
- $\eta(c) = \sum_{i=0}^{\omega-1} \eta(c_i)$;

Firstly, note that $\eta(p_0) \geq 0$. Consider $m > M = \sum_{j=0}^{\omega-1} 2^j \beta_j \geq 0$ where $0 \leq \beta_j \leq 1$ for $0 \leq j < \omega$. Then $(M \bmod 2^{i+1})$ quo $2^i = (\sum_{j=0}^{i} 2^j \beta_j)$ quo $2^i = \beta_j$. Thus, $\sum_{i=0}^{\omega-1} 2^i \eta(b_i) = \eta(p_0)$.

Now observe that $\eta(c_i) = 0$ or $2^i \eta(p_1)$, we have $0 \leq \eta(c_i) \leq 2^i \eta(p_1)$. Suppose $\eta(b_i) = 0$. Hence $\eta(c_i) = 0 \leq 2^i m \eta(b_i)$. Additionally, $2^i \eta(p_1) - 2^i m(1 - \eta(b_i)) = 2^i \eta(p_1) - 2^i m \leq 0 = \eta(c_i)$ for $\eta(p_1) < m$.

On the other hand, if $\eta(b_i) = 1$, $\eta(c_i) = 2^i \eta(p_1)$. Hence, $\eta(c_i) \geq 2^i \eta(p_1) - 2^i m(1 - \eta(b_i)) = 2^i \eta(p_1)$. Further, we have $\eta(c_i) \leq 2^i m \eta(b_i)$ for $\eta(p_1) < m$. The other constraints can be verified easily. □

Conversely, the constraints in Figure 7 indeed compute the unsigned multiplication.

Lemma 8. *Let p_0, p_1 be variables and $(p, \alpha) = \chi(p_0, p_1)$. For any valuation ρ satisfying α, $\rho(p) = \rho(p_0)\rho(p_1)$.*

Proof. We claim that for any ρ satisfies α, $\rho(c_i) = 2^i \rho(b_i)\rho(p_1)$ Suppose $\rho(b_i) = 0$. Due to the constraint $2^i m b_i \geq c_i$, we have $\rho(c_i) = 0$. On the other hand, if $\rho(b_i) = 1$, we have $\rho(c_i) \geq 2^i \rho(p_1) - 2^i m(1 - \rho(b_i)) = 2^i \rho(p_1)$. Therefore $\rho(c_i) = 2^i \rho(p_1)$ for $0 \leq \eta(c_i) \leq 2^i \eta(p_1)$.

Now $\rho(p_0) = \sum_{i=0}^{\omega-1} 2^i \rho(b_i)$. Thus, $\rho(p_0)\rho(p_1) = \sum_{i=0}^{\omega-1} 2^i \rho(b_i)\rho(p_1) = \sum_{i=0}^{\omega-1} c_i = \rho(p)$. □

Proposition 6. *Let p_0, p_1 be variables. The following statements are equivalent.*

- *There is a valuation ρ such that $0 \leq \eta(p_0) = d_0, \rho(p_1) = d_1 < m$, and $\rho(p_0)\rho(p_1) = d$;*
- *There is a valuation η such that η satisfies α, $\eta(p_0) = d_0$, $\eta(p_1) = d_1$, and $\eta(p) = d$ where $(p, \alpha) = \chi(p_0, p_1)$.*

$$\zeta(p') \triangleq \left(p, \begin{array}{c} 0 \leq a \leq 1 \\ \frac{m}{2}(a-1) \leq p' \leq \frac{m}{2}a - 1 \\ -ma \leq p + p' \leq ma \\ -m(1-a) \leq p - p' \leq m(1-a) \end{array} \right)$$

Fig. 8. Linear Constraints for Absolute Value

Proof. By Lemma 7 and 8. □

For modulo operations of terms, observe that

$$a \bmod b = a \bmod |b| = \begin{cases} |a| \bmod |b| & \text{if } a \geq 0 \\ (-|a|) \bmod |b| = |b| - (|a| \bmod |b|) & \text{if } a < 0. \end{cases}$$

We can therefore perform the modulo operations of terms by computing their absolute values first. Consider the following constraints in Figure 8 where p' has the semantic value of any term.

$$0 \leq a \leq 1$$
$$\frac{m}{2}(a-1) \leq p' \leq \frac{m}{2}a - 1$$

The variable a can only have value 0 or 1. Intuitively, p' is non-negative if and only if $a = 1$. Suppose $p' \geq 0$ and $a = 0$. We would have $-\frac{m}{2} \leq p' \leq -1$, a contradiction. Conversely, $a = 1$ implies $0 \leq p' \leq \frac{m}{2} - 1$. Hence $p' \geq 0$.

Lemma 9. *Let p' be a variable and $(p, \alpha) = \zeta(p')$. For any valuation ρ where $-\frac{m}{2} \leq \rho(p') < \frac{m}{2}$, there is a valuation η such that*

- *$\eta(p') = \rho(p')$;*
- *η satisfies α; and*
- *$\eta(p) = |\rho(p')|$.*

Proof. Define η as follows.

- $\eta(p') = \rho(p')$;
- $\eta(a) = 1$ if $\eta(p') \geq 0$; $\eta(a) = 0$ otherwise;
- $\eta(p) = \eta(p')$ if $\eta(a) = 1$; $\eta(p) = -\eta(p')$ otherwise.

Now if $\eta(p') \geq 0$, $\eta(a) = 1$. Clearly, $0 = \frac{m}{2}(\eta(a)-1) \leq \eta(p') \leq \frac{m}{2} - 1 = \frac{m}{2}\eta(a) - 1$. Furthermore, $-m\eta(a) = -m \leq \eta(p) + \eta(p') \leq m = m\eta(a)$. Also, $-m(1-\eta(a)) = 0 = \eta(p) - \eta(p') = m(1-\eta(a))$. The other case is similar. □

Lemma 10. *Let p' be a variable and $(p, \alpha) = \zeta(p')$. For any valuation ρ satisfying α, $\rho(p) = |\rho(p')|$.*

Proof. Due to the constraints $0 \leq a \leq 1$ and $\frac{m}{2}(a-1) \leq p' \leq \frac{m}{2}a - 1$, $\rho(p') \geq 0$ implies $\rho(a) = 1$ and $\rho(p) < 0$ implies $\rho(a) = 0$. If $\rho(a) = 1$, $\rho(p) - \rho(p') = 0$ by the constraints $-m(1-a) \leq p - p' \leq m(1-a)$. Hence $\rho(p) = \rho(p') = |\rho(p')|$. Similarly, $\rho(p) + \rho(p') = 0$ if $\rho(a) = 0$ by the constraints $-ma \leq p + p' \leq ma$. Therefore $\rho(p) = -\rho(p') = |\rho(p')|$. □

Proposition 7. *Let p' be a variable. The following statements are equivalent.*

- *There is a valuation ρ such that $-m \leq \rho(p') = d' \leq m$ and $|\rho(p')| = d$;*
- *There is a valuation η such that η satisfies α, $\eta(p') = d'$, and $\eta(p) = d$ where $(p, \alpha) = \zeta(p')$.*

Proof. By Lemma 9 and 10. □

We can now describe the linear constraints for non-linear terms. For term multiplication $t_0 \cdot t_1$, we first get the unsigned representation p'_0 and p'_1 of the semantic values of t_0 and t_1 respectively. This is done by the constraints $p'_0 = p_0 + ma$, $0 \leq p'_0 < m$, $p'_1 = p_1 + mb$, and $0 \leq p'_1 < m$ where $(p_0, \alpha_0) = \sigma(t_0)$ and $(p_1, \alpha_1) = \sigma(t_1)$ respectively. Then we compute the unsigned result p' by $\chi(p'_0, p'_1)$. Finally, the result is converted to the signed representation p by $p' - md = p$ and $-\frac{m}{2} \leq p < \frac{m}{2}$ (Figure 9).

To compute the semantic value of $t_0 \,\%\, t_1$, we first get the absolute values p'_0 and p'_1 of the semantic values of t_0 and t_1 by $\zeta(p_0)$ and $\zeta(p_1)$ respectively. The constraints $p'_0 - r = p'$ and $0 \leq p' < p'_1$ give $p' = |p_0| \bmod |p_1|$ where r is a multiple of $|p_1|$. Suppose $p_0 \geq 0$. Then $a = 1$ by the constraint $\frac{m}{2}(a-1) \leq p_0 \leq \frac{m}{2}a - 1$. Hence $p = p' = |p_0| \bmod |p_1|$ by the constraint $-2m(1-a) \leq p - p' \leq m(1-a)$. On the other hand, $p_0 < 0$ implies $a = 0$. Hence $p = p'_1 - p' = |p_1| - (|p_0| \bmod |p_1|)$ by the constraint $-ma \leq p - p'_1 + p' \leq 2ma$ (Figure 9).

Lemma 11. *Let t be a non-linear term in modular arithmetic and $(p, \alpha) = \sigma(t)$. For any valuation ρ, there is a valuation η such that*

- *η satisfies α;*
- *$\eta(p) = [\![t]\!]_\rho$; and*
- *$\eta(x) = \rho(x)$ for any x appearing in t.*

Proof. Let $(p_0, \alpha_0) = \sigma(t_0)$ and $(p_1, \alpha_1) = \sigma(t_1)$. There are valuations η_0 and η_1 such that $\eta_0(p_0) = [\![t_0]\!]_\rho$ and $\eta_1(p_1) = [\![t_1]\!]_\rho$ by inductive hypothesis. Let $\eta(y) = \eta_0(y)$ for all y appearing in α_0 and $\eta(y) = \eta_1(y)$ for all y appearing in α_1. Consider the following cases.

1. $t_0 \cdot t_1$. Extend η as follows.

$$\sigma(t_0 \cdot t_1) \triangleq \left(p, \begin{array}{c} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ p'_0 = p_0 + ma \\ 0 \le p'_0 < m \\ p'_1 = p_1 + mb \\ 0 \le p'_1 < m \\ p' - md = p \\ -\frac{m}{2} \le p < \frac{m}{2} \end{array} \right) \quad \text{where} \begin{array}{r} (p_0, \alpha_0) = \sigma(t_0) \\ (p_1, \alpha_1) = \sigma(t_1) \\ (p', \alpha_2) = \chi(p'_0, p'_1) \end{array}$$

$$\sigma(t_0 \,\%\, t_1) \triangleq \left(p, \begin{array}{c} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ p'_0 - r = p' \\ 0 \le p' < p'_1 \\ 0 \le a \le 1 \\ \frac{m}{2}(a-1) \le p_0 \le \frac{m}{2}a - 1 \\ -2m(1-a) \le p - p' \le m(1-a) \\ -ma \le p - p'_1 + p' \le 2ma \end{array} \right) \text{where} \begin{array}{r} (p_0, \alpha_0) = \sigma(t_0) \\ (p_1, \alpha_1) = \sigma(t_1) \\ (p'_0, \alpha_2) = \zeta(p_0) \\ (p'_1, \alpha_3) = \zeta(p_1) \\ (r, \alpha_4) = \chi(p'_1, p'') \end{array}$$

Fig. 9. Linear Constraints for Non-linear Terms

– $\eta(a) = 0$ if $0 \le \eta(p_0)$, $\eta(a) = 1$ otherwise;
– $\eta(p'_0) = m\eta(a) + \eta(p_0)$;
– $\eta(b) = 0$ if $0 \le \eta(p_1)$, $\eta(b) = 1$ otherwise; and
– $\eta(p'_1) = m\eta(b) + \eta(p_1)$.

Then $m > \eta(p'_0), \eta(p'_1) \ge 0$. Hence, there is a valuation η_2 so that η_2 satisfies α_2 and $\eta_2(p') = \eta(p'_0)\eta(p'_1)$ by Proposition 6. Now, define
– $\eta(y) = \eta_2(y)$ for all y appearing in α_2;
– $\eta(d) = \eta(p')$ squo m;
– $\eta(p) = \eta(p')$ smod m

To show $\eta(p) = [\![t_0 \cdot t_1]\!]_\rho$, observe that

$$\eta(p') \text{ smod } m = \eta(p'_0)\eta(p'_1) \text{ smod } m = \eta(p_0)\eta(p_1) \text{ smod } m.$$

Hence $[\![t_0 \cdot t_1]\!]_\rho = [\![t_0]\!]_\rho [\![t_1]\!]_\rho$ smod $m = \eta(p_0)\eta(p_1)$ smod $m = \eta(p')$ smod $m = \eta(p)$.

2. $t_0 \,\%\, t_1$. By Proposition 7, there are valuations η_2 and η_3 such that η_2 and η_3 satisfy α_2 and α_3, $\eta_2(p'_0) = |\eta(p_0)|$ and $\eta_3(p'_1) = |\eta(p_1)|$ respectively. Now, define $\eta(y) = \eta_2(y)$ for all y appearing in α_2 and $\eta(y) = \eta_3(y)$ for all y appearing in α_3. Let
– $\eta(p'') = |\eta(p_0)|$ quo $|\eta(p_1)|$;
– $\eta(p') = |\eta(p_0)|$ mod $|\eta(p_1)|$;
– $\eta(a) = 1$ if $\eta(p_0) \ge 0$, $\eta(a) = 0$ otherwise; and

– $\eta(p) = \eta(p')$ if $\eta(p_0) \geq 0$; $\eta(p'_1) - \eta(p')$ otherwise.

Since $m > \eta(p'_1), \eta(p'') \geq 0$, there is a valuation η_3 such that η_3 satisfies α_3 and $\eta_3(r) = \eta(p'_1)\eta(p'')$. Define $\eta(y) = \eta_3(y)$ for all y appearing in α_3.

To show the constraints $\frac{m}{2}(a-1) \leq p_0 \leq \frac{m}{2}a - 1$, $-2m(1-a) \leq p - p' \leq m(1-a)$, and $-ma \leq p - p'_1 + p' \leq 2ma$ are satisfied. Consider the following cases.

If $\eta(p_0) \geq 0$, $\eta(a) = 1$ and $\eta(p) = \eta(p')$. Hence $\frac{m}{2}(\eta(a)-1) = 0 \leq \eta(p_0) \leq \frac{m}{2}\eta(a) - 1$. Further, $-2m(1-\eta(a)) = 0 = \eta(p) - \eta(p') = m(1-\eta(a))$. And $-m\eta(a) = -m \leq -\eta(p'_1)$ $\leq \eta(p) - \eta(p'_1) + \eta(p') \leq 2\eta(p') \leq 2m = 0 = 2m\eta(a)$.

If $\eta(p_0) < 0$, $\eta(a) = 0$ and $\eta(p) = \eta(p'_1) - \eta(p')$. Hence $\frac{m}{2}(\eta(a)-1) = -\frac{m}{2} \leq \eta(p_0) \leq -1 = \frac{m}{2}\eta(a) - 1$. Additionally, $-2m(1-\eta(a)) = -2m \leq -2\eta(p') \leq \eta(p'_1) - 2\eta(p') = \eta(p) - \eta(p') \leq \eta(p) \leq m(1-\eta(a))$. And $-m\eta(a) = 0 = \eta(p) - \eta(p'_1) + \eta(p') = 2m\eta(a)$.

To show $\eta(p) = [\![t_0 \,\%\, t_1]\!]_\rho$, observe that $\eta(p) = |\eta(p_0)| \bmod |\eta(p_1)|$ if $\eta(p_0) \geq 0$ and $|\eta(p_1)| - (|\eta(p_0)| \bmod |\eta(p_1)|)$. □

Conversely, Figure 9 computes the semantic values of non-linear terms.

Lemma 12. *Let t be a term in non-linear modular arithmetic and $(p, \alpha) = \sigma(t)$. For any valuation ρ satisfying α, $\rho(p) = [\![t]\!]_\rho$.*

Proof. We have $-\frac{m}{2} \leq \rho(p_0) = [\![t_0]\!]_\rho, \rho(p_1) = [\![t_1]\!]_\rho < \frac{m}{2}$ by inductive hypothesis. Consider the following two cases.

1. $t_0 \cdot t_1$. Observe that $\rho(p'_0)$ and $\rho(p'_1)$ are the unsigned representations of $[\![t_0]\!]_\rho$ and $[\![t_1]\!]_\rho$ respectively. Hence $\rho(p') = \rho(p'_0)\rho(p'_1)$ by Proposition 6. Thus, $[\![t_0 \cdot t_1]\!]_\rho = [\![t_0]\!]_\rho[\![t_1]\!]_\rho$ smod m $= \rho(p'_0)\rho(p'_1)$ smod $m = \rho(p')$ smod $m = \rho(p)$.
2. $t_0 \,\%\, t_1$. We have $\rho(p'_0) = |\rho(p_0)| = |[\![t_0]\!]_\rho|$ and $\rho(p'_1) = |[\![t_1]\!]_\rho|$ by Proposition 7. Also, $\rho(p')$ $= \rho(p'_0) \bmod \rho(p'_1) = |[\![t_0]\!]_\rho| \bmod |[\![t_1]\!]_\rho|$.
   If $\rho(a) = 1$, $\rho(p_0) \geq 0$ by the constraint $\frac{m}{2}(a-1) \leq p_0$. And $\rho(p) = \rho(p') = |[\![t_0]\!]_\rho| \bmod |[\![t_1]\!]_\rho|$ by the constraint $-2m(1-a) \leq p - p' \leq m(1-a)$. Hence $[\![t_0 \,\%\, t_1]\!]_\rho = |[\![t_0]\!]_\rho| \bmod |[\![t_1]\!]_\rho| = \rho(p') = \rho(p)$.
   If $\rho(a) = 0$, $\rho(p_0) < 0$ by the constraint $p_0 \leq \frac{m}{2}a - 1$. And $\rho(p) = \rho(p'_1) - \rho(p') = |[\![t_1]\!]_\rho| - (|[\![t_0]\!]_\rho| \bmod |[\![t_1]\!]_\rho|)$ by the constraint $-ma \leq p - p'_1 + p' \leq 2ma$. Thus, $[\![t_0 \,\%\, t_1]\!]_\rho$ $= |[\![t_1]\!]_\rho| - (|[\![t_0]\!]_\rho| \bmod |[\![t_1]\!]_\rho|) = \rho(p'_1) - \rho(p') = \rho(p)$. □

Proposition 8. *Let t be a non-linear term in modular arithmetic. The following statements are equivalent.*

- *There is a valuation ρ such that $[\![t]\!]_\rho = d$.*
- *There is a valuation η such that η satisfies α and $\eta(p) = d$ where $(p, \alpha) = \sigma(t)$.*

Proof. By Lemma 11 and 12. □

Theorem 3. *Given a formula f in modular arithmetic over $\mathbb{Z}_m$ where $m = 2^\omega$, the satisfiability problem can be solved by an instance of the integer programming problem with the number of constraints and variables linear in $\omega|f|$.*

Proof. For each term multiplication, $6\omega + 12$ constraints and $2\omega + 7$ variables are added. For each modulo operation of terms, $6\omega + 26$ constraints and $2\omega + 7$ variables are needed (4 of the constraints and one variable in $\zeta(p_0)$ are shared). Since there are at most $|f|$ term multiplications and modulo operations in f, the number of constraints and variables is $O(\omega|f|)$. □

6 Applications

Since modular arithmetic is the default integral computation in conventional programming languages, our decision procedure may be useful in software verification. For hardware verification, our reduction may work as a non-linear constraint solver which accepts control signals from other decision procedures. Particularly, we find that the following areas may benefit from our algorithm.

Modern proof assistants allow external decision procedures to discharge proof obligations [31, 16]. Although modular arithmetic is essential to many number theoretic and cryptographic algorithms, there is no proof assistant which provides decision procedures for modular arithmetic to the best of our knowledge. Since it is rather tedious to deal with modular arithmetic in each integral computation, verifiers simply assume the infinite-precision integer model in program verification. Subsequently, algorithms certified by proof assistants are not exactly the same as their implementations. Our procedure may help verifiers work in a more realistic computational model.

If a proof assistant is used to determine the truth values of predicates, the abstract model may be inadequate in predicate abstraction [10, 28] for the same reason. In the presence of modular arithmetic, our integer programming-based procedure may be more efficient than, say, SAT-based predicate abstraction [7, 8]. The new technique will refine the abstraction and may perform better in such circumstances.

Another possible application of our algorithm is SAT-based model checking [4, 1, 32]. Our word-level decision procedure may be better for models with modular arithmetic, but it does not seem to fare well on Boolean satisfiability. However, modern integer programming packages support distributed computation. Thanks to the geometric interpretation, parallel integer programming optimizer is now available [26]. Our approach gives a parallel SAT solver indirectly.

7 Conclusion

Deciding the satisfiability of modular arithmetic formula is essential in software verification. We have characterized the complexity of its satisfiability problem and provided an efficient reduction to integer programming problem. Our result shows that it is more efficient to develop specialized algorithms than apply a more general algorithm for Presburger arithmetic. Additionally, the number of constraints and variables is linear in the length of the input formula in our reduction. With heuristics like relaxation and rounding, the satisfiability problem could be solved efficiently by modern integer programming packages in practice.

It would be interesting to compare our algorithm with other techniques [5, 29, 2], especially those with the binary encoding scheme. Since the satisfiability problem of modular arithmetic formula is **NP**-complete, one could also build a decision procedure based on SAT solvers. But the binary encoding would eliminate the mathematical nature of the problem. It is unclear which approach will prevail in practice.

References

1. Awedh, M., Somenzi, F.: Proving more properties with bounded model checking. In Alur, R., Peled, D.A., eds.: Computer Aided Verification. Volume 3114 of LNCS., Springer Verlag (2004) 96–108
2. Babić, D., Musuvathi, M.: Modular arithmetic decision procedure. Technical Report MSR-TR-2005-114, Microsoft Research (2005)
3. Ball, T., Rajamani, S.K.: The SLAM toolkit. In Berry, G., Comon, H., Finkel, A., eds.: Computer Aided Verification. Volume 2102 of LNCS., Springer-Verlag (2001) 260–264
4. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: Design Automation Conference, New York, ACM Press (1999) 317–320
5. Boigelot, B., Wolper, P.: Representing arithmetic constraints with finite automata: An overview. In Stuckey, P.J., ed.: International Conference on Logic Programming. Volume 2401 of LNCS., Springer-Verlag (2002) 1–19
6. Brinkmann, R., Drechsler, R.: RTL-datapath verification using integer linear programming. In: ASP-DAC/VLSI Design, IEEE Computer Society (2002) 741
7. Chaki, S., Clarke, E., Groce, A., Jha, S., Veith, H.: Modular verification of software components in C. In Dillon, L., Tichy, W., eds.: International Conference on Software Engineering, IEEE Computer Society (2003) 385–395
8. Clarke, E., Kroening, D., Sharygina, N., Yorav, K.: Predicate abstraction of ANSI-C programs using SAT. Formal Methods in System Design **25**(2–3) (2004) 105–127
9. Cooper, D.C.: Theorem proving in arithmetic without multiplication. Machine Intelligence **7** (1972)
10. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: ACM Symposium on Principles of Programming Languages. (1977) 238–252
11. Dantzig, G.B.: Linear Programming and Extensions. 10 edn. Princeton University Press (1993)
12. Enderton, H.: A Mathematical Introduction to Logic. Academic Press (1972)
13. Granger, P.: Static analysis of linear congruence equalities among variables of a program. In Abramsky, S., Maibaum, T.S.E., eds.: Theory and Practice of Software Ddevelopment. Volume 439 of LNCS., Springer-Verlag (1991) 169–192
14. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Software verification with Blast. In Ball, T., Rajamani, S.K., eds.: 10th SPIN Workshop on Model Checking Software. Volume 2648 of LNCS., Springer-Verlag (2003) 235–239

15. Holzmann, G.: The model checker SPIN. IEEE Transaction on Software Engineering **23**(5) (1997) 279–295
16. Huet, G., Kahn, G., Paulin-Mohring, P.M.: The Coq proof assistant: a tutorial: version 6.1. Technical Report 204, Institut National de Recherche en Informatique et en Automatique, Le Chesnay, France (1997)
17. Hungerford, T.W.: Algebra. Volume 73 of Graduate Texts in Mathematics. Springer-Verlag (1980)
18. IEEE: IEEE Standard for the Scheme Programming Language. (1991)
19. Knuth, D.E.: The Art of Computer Programming. Volume II, Seminumerical Algorithms. Addison-Wesley (1997)
20. Müller-Olm, M., Seidl, H.: Analysis of modular arithmetic. In Sagiv, M., ed.: Programming Languages and Systems. Volume 3444 of LNCS., Springer-Verlag (2005) 31–45
21. Müller-Olm, M., Seidl, H.: A generic framework for interprocedural analysis of numerical properties. In Hankin, C., Siveroni, I., eds.: Satatic Analysis. Volume 3672 of LNCS., Springer-Verlag (2005) 235–250
22. Oppen, D.C.: Elementary bounds for presburger arithmetic. In: Proceedings of the fifth annual ACM Symposium on Theory of Computing, ACM (1973) 34–37
23. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
24. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Inc (1982)
25. Pugh, W.: The omega test: a fast and practical integer programming algorithm for dependence analysis. In Martin, J.L., ed.: 1991 ACM/IEEE conference on Supercomputing, ACM Press (1991) 4–13
26. Ralphs, T.K., Guzelsoy, M.: The SYMPHONY callable library for mixed integer programming. In: INFORMS Computing Society. (2005)
27. Reddy, C.R., Loveland, D.W.: Presburger arithmetic with bounded quantifier alternation. In: Proceedings of the tenth annual ACM Symposium on Theory of Computing, ACM (1978) 320–325
28. Saídi, H., Graf, S.: Construction of abstract state graphs with PVS. In Grumberg, ed.: Computer Aided Verification. Volume 1254 of LNCS., Springer Verlag (1997) 72–83
29. Seshia, S.A., Bryant, R.E.: Deciding quantifier-free presburger formulas using parameterized solution bounds. In: Logic in Computer Science, IEEE Computer Society (2004) 100–109
30. Stinson, D.R.: Cryptography: Theory and Practice. CRC Press, Inc (1995)
31. T.F. Melham: Introduction to the HOL theorem prover. University of Cambridge, Computer Laboratory, Cambridge, England. (1990)
32. Wang, B.Y.: Proving $\forall\mu$-calculus properties with SAT-based model checking. In Wang, F., ed.: Formal Techniques for Networked and Distributed Systems. Volume 3731 of LNCS., Springer-Verlag (2005) 113–127