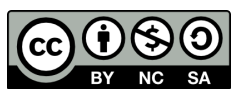
CDM

Presburger Arithmetic

K. Sutner

Carnegie Mellon University

Spring 2025




A80501-66
SX837
intel®
pentium™
PROCESSOR
L4142580
INTEL©1992

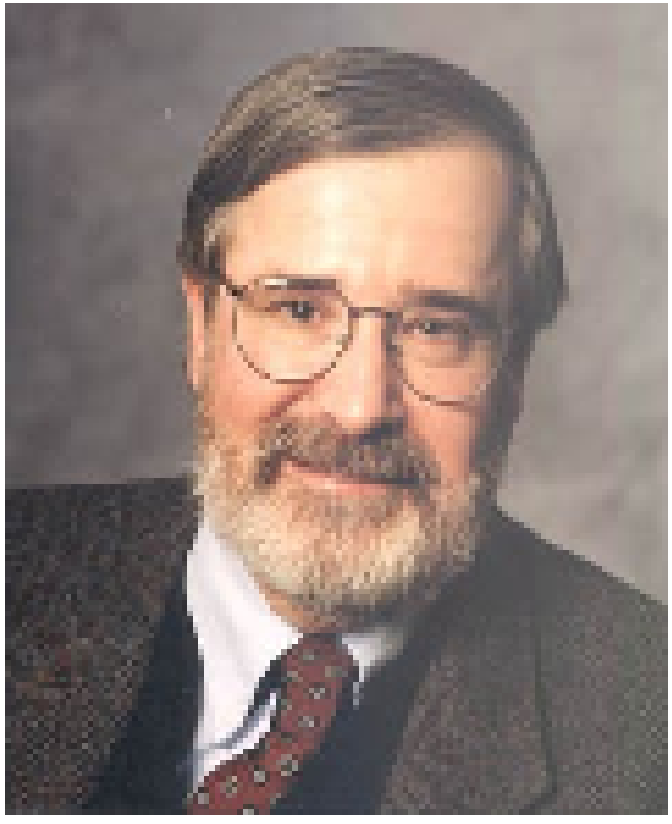
$$4195835.0/3145727.0 = 1.33382044913624100\text{25} \quad \text{correct}$$
$$4195835.0/3145727.0 = 1.333\text{7390689020375894} \quad \text{pentium}$$

Alternatively

$$4195835.0 - 3145727.0 * (4195835.0/3145727.0) = 0 \quad \text{correct}$$
$$4195835.0 - 3145727.0 * (4195835.0/3145727.0) = \text{256} \quad \text{pentium}$$

This fatal bug was discovered in October 1994 by number theorist Thomas R. Nicely, doing research in pure math. Intel did not respond well, and lost around $500 mill.

Incidentally, this is a software problem at heart.



If applied, Clarke's model checking methods would have found the bug.

More importantly, they were powerful enough to **prove** that the patch Intel concocted was indeed correct.


ariane 502
esa
esa
cnes
cnes

On June 4, 1996, the maiden flight of the European Ariane 5 rocket crashed about 40 seconds after takeoff. The direct financial loss was about half a billion dollars, all uninsured. An investigation showed that the explosion was the result of a software error.

Particularly annoying is the fact that the problem came from a piece of the software that was not needed during the crash, the Inertial Reference System. Before lift-off certain computations are performed to align the IRS. Normally they should be stopped at -9 seconds. However, a hold in the countdown would require a reset of the IRS taking several hours.

To avoid this nuisance, the computation was allowed to continue even after the system had switched to flight mode.

In the Ariane 5, this caused an uncaught exception due to a floating-point error: a conversion from a 64-bit integer (which should have been less than 2^{15}) to a 16-bit signed integer was erroneously applied to a greater number: the "horizontal bias" of the flight depends much on the size of rocket–and Ariane 5 was larger than its predecessors.

There was no explicit exception handler (since it presumably was not needed), so the entire software crashed and the launcher with it.

Suppose you are implementing a dynamic programming algorithm that has to fill out an $n \times n$ array X. The algorithm

- initializes the first row and the first column,
- then fills in the whole array according to

$$X[i, j] = \text{func}(X[i-1, j], X[i, j-1])$$

- lastly, reads off the result in $X[n-1, n-1]$.

We would like to check that all the array access operations are safe, and that the result is properly computed.

And, we want to do so automatically.

One of the problems is that the recurrence can be implemented in several ways:

```
// column by column
for i = 1 .. n-1 do
for j = 1 .. n-1 do
      X[i,j] = func( X[i-1,j], X[i,j-1] )


// row by row
for j = 1 .. n-1 do
for i = 1 .. n-1 do
      X[i,j] = func( X[i-1,j], X[i,j-1] )


// by diagonal
for d = 1 .. 2n-3 do
for i = 1 .. d do
      X[i,d-i+1] = func( X[i-1,d-i+1], X[i,d-i] )
```

For a human, it is easy to see that the row-by-row and column-by-column methods are correct.

The diagonal approach already requires a bit of thought: why $2n - 3$?

The good news is that the index arithmetic involved in the algorithms is quite simple, essentially all we need is addition and order. That is very fortunate, since arithmetic in general is a tough nut to crack.

Question:
Can we get away with some decidable "baby arithmetic"?

Theorem (Y. Matiyasevic, 1970)

It is undecidable whether a Diophantine equation has a solution in the integers.

Suppose we wanted to build a reasoning system for ordinary Dedekind-Peano Arithmetic (DPA). So we are trying to axiomatize the structure

$$\mathfrak{N} = \langle \mathbb{N}, +, *, 0, 1; < \rangle$$

Dedekind and Peano figured out how to do this in 1888/1889, respectively. Similar axiomatizations can be found for $\mathbb{Z}$.

The main idea is that both addition and multiplication come down to induction (recursion) applied to the successor function $x \mapsto x + 1$.

But there is a problem: the terms in (DPA) are essentially multivariate polynomials with integral coefficients:

$$t(\boldsymbol{x}) = \sum_{\boldsymbol{e}} c_{\boldsymbol{e}}\, \boldsymbol{x}^{\boldsymbol{e}}$$

where $\boldsymbol{x}^{\boldsymbol{e}} = x_1^{e_1} x_2^{e_2} \ldots x_k^{e_k}$.

So a decision algorithm would have to deal in particular with sentences of the form

$$\exists\, \boldsymbol{x}\, \big(t(\boldsymbol{x}) = 0\big)$$

Matiyasevic's celebrated theorem shows that this is already undecidable. Even the existential part of arithmetic is undecidable, never mind assertions with multiple quantifiers.

And Worse

Never mind more complicated assertions with alternating quantifiers and the like:

$$\forall\, x\, \exists\, u, v\, \Big(\mathsf{even}(x) \wedge x \geq 4 \Rightarrow x = u + v \wedge \mathsf{prime}(u) \wedge \mathsf{prime}(v)\Big)$$

$$\forall\, x\, \exists\, y\, \Big(x < y \wedge \mathsf{prime}(y) \wedge \mathsf{prime}(y + 2)\Big)$$

$$\mathsf{even}(x) \Leftrightarrow \exists\, u\, \Big(x = u + u\Big)$$

$$\mathsf{prime}(x) \Leftrightarrow 2 \leq x \wedge \forall\, u, v\, \Big(x = u * v \Rightarrow u = 1 \vee v = 1\Big)$$

If we want to decide sentences in arithmetic, we need to use something weaker than Dedekind-Peano arithmetic. Here is a radical proposal: let's just forget about multiplication.

Here is a radical proposal: let's just forget about multiplication. More precisely, we restrict ourselves to the structure

$$\mathfrak{N}_- = \langle \mathbb{N}, +, 0, 1; < \rangle$$

As far as first-order logic is concerned, this structure is actually a bit overly verbose, we could get away with just $\langle \mathbb{N}, + \rangle$. To see why, note that we can define all the rest:

$$\begin{array}{ll} x = 0 & \forall\, z\, (x + z = z) \\ x \leq y & \exists\, z\, (x + z = y) \\ x < y & x \leq y \wedge x \neq y \\ x = 1 & 0 < x \wedge \forall\, z\, (0 < z \Rightarrow x \leq z) \\ x = y + 1 & y < x \wedge \forall\, z\, (y < z \Rightarrow x \leq z) \end{array}$$

Full multiplication is absent, but multiplication by a **constant** is available; for example

$$y = 3 * x \iff y = x + x + x$$

We can also do modular arithmetic with **fixed modulus**:

$$y = x \bmod 3 \iff \exists\, z\, (x = 3 * z + y \wedge y < 3)$$
$$y = x \operatorname{div} 3 \iff \exists\, z\, (x = 3 * y + z \wedge z < 3)$$

This may seem pretty feeble and it does turn out to be the case that sets of numbers definable in Presburger arithmetic are fairly simple. Alas, finding a decision algorithm is not so easy.

The terms in this language are pretty weak, no more than affine combinations:

$$t(\boldsymbol{x}) = c + \sum c_i x_i$$

where all the c, c_i are constant, $c, c_i \in \mathbb{N}$.

This is easy to check by induction on terms.

In particular, we have lost multivariate polynomials–which is a good thing, since otherwise Matiyasevic's theorem would automatically doom any attempt at finding a decision algorithm.

Definition

A set $A \subseteq \mathbb{N}$ is linear if $A = \{ c + \sum c_i x_i \mid x_i \geq 0 \}$.
Here c is the constant and the c_i are the periods.

A set $A \subseteq \mathbb{N}$ is semilinear if it is the finite union of linear sets.

Intuitively, the semilinear sets are the ultimately periodic subsets of $\mathbb{N}$.

Actually, this is just the one-dimensional case; one can generalize easily to subsets of $\mathbb{N}^d$ by letting $c, c_i \in \mathbb{N}^d$.

Note that every finite set is semilinear.

Lemma

Semilinear sets are closed under union, intersection and complement.

The main idea is that $A = \{\, c + \sum c_i x_i \mid x_i \geq 0 \,\}$ can be rewritten in the form

$$A = F \cup \{\, c + d\, x \mid x \geq 0 \,\}$$

where $d = \gcd(c_1, \ldots, c_n)$.

The complement of A is then clearly semilinear.

Exercise

Figure out the details.

Clearly, semilinear sets are definable in Presburger Arithmetic:

$$z \in A \iff \exists \boldsymbol{x} \left(t_1(\boldsymbol{x}) = z \vee t_2(\boldsymbol{x}) = z \vee \ldots \vee t_k(\boldsymbol{x}) = z \right)$$

Theorem (Ginsburg, Spanier 1964)

The sets definable in Presburger Arithmetic are exactly the semilinear sets.

This is rather surprising: the formula above is purely existential; your intuition might tell you that more complicated quantifier structures would produce more complicated sets.

Suppose we code natural numbers in unary: $n \mapsto a^n$.

Then every set $A \subseteq \mathbb{N}$ corresponds to a tally language $L_A \subseteq \{a\}^\star$.

Theorem

A set $A \subseteq \mathbb{N}$ is semilinear if, and only if, L_A is regular.

This follows directly from the fact that DFAs over a one-letter alphabet all have the shape of a lasso.

It also provides a proof that semilinear sets form a Boolean algebra.

This is one example where a characterization in terms of finite state machines is (mildly) useful.

The previous results suggest that Presburger arithmetic should be fairly simple. So one might be tempted to take on the following challenge:

> Find a decision algorithm for Presburger arithmetic.

Here is a fairly simple formula in Presburger arithmetic:

$$\Phi \equiv \exists\, u \,\forall\, v \left(u < v \Rightarrow \exists\, x, y \,(3x + 5y = v)\right)$$

How would a decision algorithm tackle this formula?

A human having suffered through 21128/15151 knows the answer is: true, since 3 and 5 are coprime, but the proof requires a bit of basic number theory.

Worse, the problem naturally generalizes to several multipliers. One wants to compute the largest number v such that

$$v = \boldsymbol{a} \circ \boldsymbol{x} = a_1 x_1 + a_2 x_2 + \ldots + a_d x_d$$

has no solution $\boldsymbol{x} \in \mathbb{N}^d$.

This is often written as the Frobenius function $g(a_1, \ldots, a_d)$.

We can easily define the Frobenius function in Presburger arithmetic:

$$g(\boldsymbol{a}) = b \iff \neg\exists\, \boldsymbol{x}\, (b = \boldsymbol{a} \circ \boldsymbol{x}) \wedge\ \forall\, z > b\, \exists\, \boldsymbol{x}\, (z = \boldsymbol{a} \circ \boldsymbol{x})$$

So What?

The only easy proposition about the Frobenius function is the 2-dimensional case:

$$g(a_1, a_2) = a_1 a_2 - a_1 - a_2$$

The general case is notoriously hard. E.g., it is $\mathbb{NP}$-hard to compute $g(\boldsymbol{a})$. There is a polynomial time algorithm when d is fixed, though.

Here is the rub: Any general decision algorithm would provide at least some sort of insight into to the Frobenius function. So, there is really no hope that such an algorithm could be simple.

Presburger used an important method called quantifier elimination: by transforming a formula into another, equivalent one that has one fewer quantifiers. Ultimately, we can get rid of all quantifiers.

The remaining quantifier-free formula is equivalent to the original one, and is easily tested for validity.

So a single step takes us from

$$\Phi = \exists\, x_1 \,\forall\, x_2 \;\ldots Q_n\, x_n \underbrace{\exists\, z}_{\text{kill}} \,\varphi(z, \boldsymbol{x})$$

to an equivalent formula

$$\Phi' = \exists\, x_1 \,\forall\, x_2 \;\ldots Q_n\, x_n \,\varphi'(\boldsymbol{x})$$

where the elimination variable z is no longer present.

Universal quantifiers are handled via $\forall\, x \;\equiv \neg\, \exists\, x \;\neg$.

Example

Suppose we want to eliminate $\exists\, z$ in

$$2x + 4y - 3z < 7 \quad \wedge \quad 3x - y + 2z < -4$$

Rearrange to get an interval

$$2x + 4y - 7 < 3z \quad \wedge \quad 2z < -3x + y - 4$$

Scale the z terms

$$4x + 8y - 14 < 6z \quad \wedge \quad 6z < -9x + 3y - 12$$

Over $\mathbb{Q}$, this is equivalent to

$$0 < -13x - 5y + 2$$

Unfortunately, the last step, while correct over $\mathbb{Q}$, does not work over $\mathbb{Z}$.

There we have to make sure that there is a number divisible by 6 in the interval.

To handle this problem, we enlarge our domain from $\mathbb{N}$ to $\mathbb{Z}$: this is fine, we can recover the naturals by adding constraints $0 \leq x$.

Next we expand our language slightly (this is critical, QE fails for the original language).

- Add a constant c for each integer.
- Add the subtraction operation $x - y$.
- Add a multiplication function $x \mapsto c * x$ for each integer c.
- Add a divisibility predicate $c \mid x$ for each integer c.

We won't introduce special symbols for the new multiplication function and divisibility predicate.

Nothing New

The key point is that these extensions do not change the expressiveness of our system, we can always translate back into the original Presburger language.

$$x - y = z \qquad x = y + z$$
$$3 * x = z \qquad x + x + x = z$$
$$3 \mid x \qquad \exists z\,(3 * z = x)$$

Note the hidden existential quantifier in the divisibility relation. As a consequence, we can recursively remove quantifiers in the new language.

Note that all terms in the extended language are still linear multinomials.

We would like to construct suitable terms t_i that do not contain z such that

$$\exists\, z\, \varphi(z, \boldsymbol{x}) \iff \varphi(t_1, \boldsymbol{x}) \vee \varphi(t_2, \boldsymbol{x}) \vee \ldots \vee \varphi(t_k, \boldsymbol{x})$$

Since an existential quantifier is a kind of disjunction, this is not entirely perplexing. Of course, we need a finite disjunction even when the domain is infinite.

Note that $\Leftarrow$ holds automatically, but $\Rightarrow$ requires work.

This trick is often used in quantifier elimination.

Cleanup

We want our formulae to have a simple, uniform syntactic structure. To this end, we eliminate equalities, non-equalities and non-divisibility assertions.

$$
\begin{aligned}
t_1 = t_2 \quad &\leadsto \quad t_1 < t_2 + 1 \wedge t_2 < t_1 + 1 \\
t_1 \neq t_2 \quad &\leadsto \quad t_1 < t_2 \vee t_2 < t_1 \\
\neg(t_1 < t_2) \quad &\leadsto \quad t_2 < t_1 + 1 \\
\neg(c \mid t) \quad &\leadsto \quad \bigvee_{i=1}^{c-1} c \mid t + i
\end{aligned}
$$

Logically these rewrite steps are quite straightforward, but note that the size of the formula increases. E.g., the first rule more or less doubles the size.

We may safely assume that φ is in disjunctive normal form. Since

$$\exists\, z \left(\varphi_1(z) \vee \varphi_2(z)\right) \iff \exists\, z\, \varphi_1(z) \vee \exists\, z\, \varphi_2(z)$$

it suffices to remove the quantifier from each of the conjuncts $\psi = \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k$. Note that all the ψ_i are just atomic formulae.

We may also assume that z actually occurs in ψ_i since

$$\exists\, z \left(\psi_1(z, \boldsymbol{x}) \wedge \psi_2(\boldsymbol{x})\right) \iff \exists\, z\, \psi_1(z, \boldsymbol{x}) \wedge \psi_2(\boldsymbol{x})$$

So we are left with a conjunction of atomic formulae that all contain z, plus some other variables $\boldsymbol{x}$.

We can rearrange things a bit to get a formula that looks like

$$\Psi(z) = \bigwedge r_i < a_i z \ \wedge \ \bigwedge b_i z < s_j \ \wedge \ \bigwedge c_k \mid \left(d_k z + t_k\right)$$

where the r_i, s_i and t_i are all terms (usually containing $\boldsymbol{x}$), but the the a_i, b_i, c_i and d_i are constants.

So we have lower bounds, upper bounds and divisibility constraints, and we have to determine whether there is an integer z that fits the bill.

Here is a strange trick to further simplify the bound-divisibility formula.

Inane Observation:

$$s < t \iff ks < kt \qquad\qquad c \mid t \iff kc \mid kt$$

for all positive k.

Now let μ be the least common multiple of all the a_i, b_i and c_i. By multiplying each of the bound-divisibility terms, we can make sure that the coefficient of z everywhere is $\pm\mu$.

Hence we can substitute a new variable Z for $\mu\, z$ and we get the temporary equivalent formula

$$\exists\, Z \left(\Psi_0(Z) \wedge \mu \mid Z\right)$$

It may look like we have made things worse in Φ, but note that the coefficients of Z are now ± 1.

Hence, we can rearrange the terms in Ψ_0 one more time to get

$$\Psi'(z) = \bigwedge r_i < z \ \wedge \ \bigwedge z < s_j \ \wedge \ \bigwedge c_k \mid (z + t_k)$$

We can add the divisibility constraint $\mu \mid Z$ to the last conjunction, so Ψ' already is equivalent to our temporary expression.

If we ignore the divisibility constraints, then this is equivalent to

$$\bigwedge_{ij} r_i + 1 < s_j$$

and we have successfully eliminated z.

Assume there is at least one lower bound. Then the following z-free formula Φ is equivalent to $\exists\, z\, \Psi(z)$:

$$\Phi = \bigvee_i \bigvee_k \Psi'(r_i + k)$$

where the index k is supposed to range up to C, the least common multiple of all the divisors c_i in Ψ'.

If there are no lower bounds on z, we can ignore the upper bounds, too. Divisibility can still be handled by

$$\Phi = \bigvee_k \Psi'(k)$$

Example: Frobenius

$$\begin{aligned}\Phi &= \exists\, u \,\forall\, v \,\exists\, x, y \left(u < v \Rightarrow 3x + 5y = v\right) \\ \Psi(z) &= u < v \Rightarrow 3x + 5z = v \\ &\equiv \neg(u < v) \vee \left(v < 3x + 5z + 1 \wedge 3x + 5z < v + 1\right)\end{aligned}$$

Rearrange for upper/lower bounds, drop z-free part

$$\left(v - 3x - 1 < 5z \wedge 5z < v - 3x + 1\right)$$

Expose and rescale the variable

$$\left(v - 3x - 1 < z \wedge z < v - 3x + 1 \wedge 5 \mid z\right)$$

There is one lower/upper bound and one divisibility constraint.

Eliminate the variable via a disjunction:

$$\bigwedge_{\ell \in [5]} \big(v - 3x - 1 < v - 3x - 1 + \ell \wedge v - 3x - 1 + \ell < v - 3x + 1 \wedge 5 \mid v - 3x - 1 + \ell\big)$$

With very basic simplifications of inequalities this turns into

$$\bigwedge_{\ell \in [5]} \big(\top \wedge \ell < 2 \wedge 5 \mid v - 3x - 1 + \ell\big)$$

and then to $5 \mid v - 3x$.

This leaves the equivalent formula

$$\Phi = \exists\, u \,\forall\, v \,\exists\, x \,\big(u < v \Rightarrow 5 \mid v - 3x\big)$$

This time we work on

$$\begin{aligned}\Psi(z) &= 5 \mid v - 3z \\ &\equiv 5 \mid v - z \wedge 3 \mid z \\ &\equiv \bigvee_{\ell \in [15]} \big(3 \mid \ell \wedge 5 \mid v - \ell\big)\end{aligned}$$

Again we need some minor simplification mechanism to simplify this to

$$\bigvee_{\ell \in D} \big(5 \mid v - \ell\big) \qquad\qquad D = \{3, 6, 9, 12, 15\}$$

This leaves the equivalent formula

$$\Phi = \exists\, u \forall\, v \left(u < v \Rightarrow \bigvee_{\ell \in D} \big(5 \mid v - \ell\big) \right)$$

This time we have deal with a universal quantifier, so we need to convert to an existential one plus two negations.

$$\Phi = \exists\, u\, \neg \exists\, v \left(u < v \wedge \bigwedge_{\ell \in D} \left(5 \not| \, v - \ell \right) \right)$$

We have to remove the negation in front of divisibility.

$$\Phi = \exists\, u\, \neg \exists\, v \left(u < v \wedge \bigwedge_{\ell \in D} \bigvee_{i \in [4]} \left(5 \mid v - \ell + i \right) \right)$$

The matrix is now in CNF but we need DNF.

In this case, this comes down to "multiplying out" and produces a much longer formula: a priori there are $4^5 = 1024$ terms in the DNF.

Again, we resort to algebraic simplification to avoid insanely large expressions.

The condition really reads

$$\bigwedge_{\ell \in D} \bigvee_{i \in [4]} \big(v = \ell - i \bmod 5\big)$$

But $D \bmod 5 = \{3, 1, 4, 2, 0\}$, so for every v there is some $\ell \in D$ such that $v - \ell = 0 \bmod 5$. Hence the whole formula simplifies to $\bot$.

The second negation turns this into $\top$ and the last quantifier falls by the wayside: the formula is true.

How about the formula

$$\Phi = \exists\, u \,\forall\, v \,\exists\, x, y \left(u < v \Rightarrow 6x + 10y = v \right)$$

The algorithm works along the same lines as in the last example, except that this time wind up with

$$D = \{6, 12, 18, 24, 30\}$$

and the critical assertion

$$\exists\, v \left(u < v \wedge \bigwedge_{\ell \in D} \bigvee_{i \in [9]} \left(10 \mid v - \ell + i \right) \right)$$

turns out to be true (in fact $D = \{6, 12\}$ would suffice). Negation turns this into false, and the whole formula is invalid.

- In 1929, Presburger showed that Dedekind-Peano arithmetic without multiplication (Presburger arithmetic) is decidable.
- In 1930, Skolem proved that Dedekind-Peano arithmetic without addition (Skolem arithmetic) is decidable.
- In 1931, Gödel showed that full Dedekind-Peano arithmetic (actually: any reasonable arithmetic) is incomplete and hence necessarily undecidable.
- In 1948, Tarski showed that real arithmetic is decidable.
- In 1970, Matiyasevic showed that Diophantine equations are undecidable.

Once decidability is established, the next question is efficiency: can the algorithm be used for potential applications?

It should be clear from the quantifier elimination process given here that an implementation is somewhat messy: the number of rounds is the same as the number of variables, but the formulae grow substantially.

As stated, the algorithm is not elementary (no fixed-size stack of 2s will do): e.g., the conversion from CNF to DNF could blow up exponentially in earch round.

The good news is that, with more effort, one can get the worst-case running time down to "just" triple-exponential

$$O\left(2^{2^{2^{cn}}}\right)$$

On the other hand, it turns out that there is no algorithm, regardless of the approach it might be based on, that is better than doubly exponential. Of course, the lower bounds are based on somewhat artificially constructed inputs.

$$\Omega\left(2^{2^{cn}}\right)$$

So the real challenge is to find an algorithm that works reasonably well on inputs that actually appear in applications such as theorem proving and program verification.