



Algebraic Indexing to Parallelize Atom Comparison in the ATP Saturation Loop

Theodore Kim and Adam Pease

Naval Postgraduate School, Monterey, CA, USA
theodore.kim@nps.edu, adam.pease@nps.edu

Abstract

This paper introduces a scheme to compare atoms and extract variable substitution bindings, as a part of unification, within the core loop of an automated theorem prover. The presented algorithm uses only linear algebraic transformations and no pointer operations and is designed to scale with parallelized hardware acceleration. Atoms are converted to a single 64-bit value using a fingerprint function, under which identity comparison is a single XOR and unifiability is dictated by membership in a linear span. Two mechanisms follow, blackout matching (reduced-basis membership) and power unification (recovering variable bindings by solving a linear system in $GF(2^{64})$). Although the comparison is approximate, it serves as a prefilter whose candidate matches can be verified syntactically before inference. When correctly tuned, it achieves substantial candidate reduction during a simulated unification pass on randomly selected axioms with zero observed false negatives in targeted ground/pattern and pattern/pattern validation.

1 Introduction

Modern computational performance is often tied to the utilization of hardware parallelism. The architectures of automated theorem provers (ATP), particularly their methods of indexing and comparing atoms, have limited scalability with the concurrency provided by hardware parallelism. Rather than index and unify terms via a sequential, memory-addressed tree walk, a unification algorithm could instead identify a term in a single pass by treating each atom as a single 64-bit value and comparing them directly using algebra in $GF(2^{64})$.

The goal of this paper is to define a **fingerprint function** (id) such that if two relations are equivalent their fingerprints are equal; the term “fingerprint” follows Schulz [1] as a method of identifying atoms. This paper describes this method in Section 3.

$$R \equiv Q \implies \text{id}(R) = \text{id}(Q) \quad (1)$$

Furthermore, such a fingerprinting scheme should also make possible some filter ϕ that can be performed on two atoms’ encoded identities to imply whether the two can be unified through some substitution operation. This operation is outlined in Section 4.

$$\phi : \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{B} \quad \forall R, Q. \neg \phi(\text{id}(R), \text{id}(Q)) \implies \neg \exists \sigma. R\sigma \equiv Q\sigma \quad (2)$$

Finally, the fingerprint should lend itself not just to an indication of a unifiable substitution but also to an operation which extracts the actual substitution bindings. Let $\mathbf{id}^{(k)}(R) = (\text{id}_1(R), \dots, \text{id}_k(R))$ denote the vector of k fingerprints of R . This process is defined in Section 5. Importantly, these goals are to be fulfilled without the use of branches or memory operations aside from loading arrays of candidate atoms to and from memory.

$$\exists g. g : \mathbb{F}^k \times \mathbb{F}^k \rightarrow \mathbb{F}^k. \sigma = \{v_\ell \mapsto t_\ell\} \wedge R\sigma \equiv Q \implies g(\mathbf{id}^{(k)}(R), \mathbf{id}^{(k)}(Q)) = (\text{id}(t_1), \dots, \text{id}(t_k)) \quad (3)$$

This paper includes a discussion of the necessary background to support the claims made here (Section 2) and an empirical verification of the mathematical claims made in support of its various mechanisms (Section 6). Importantly, this paper describes only the design of the fingerprinting function and the two relevant comparison operations which also enable variable binding extraction. Section 7 introduces the application of the methodologies discussed here in the implementation of an ATP, as well as some further, potential applications.

2 Background

2.1 Automated Theorem Proving

ATPs prove logical conjectures by negating the conjecture and attempting to find a contradiction using the provided axiom set. The given-clause algorithm of Otter and E [2, 3] involves two stacks of clauses, the active and passive stacks. From the active stack, the ATP selects a given clause and performs **unification** [4] against clauses in the passive stack that contain a literal of opposite polarity, deriving new clauses and placing them on the active stack. Deriving the empty clause is indicative of a contradiction and the theorem is proven. Provers are said to be sound if they never produce a proof of a non-theorem. A unification algorithm is perfect if it never claims existence of a substitution that cannot transform two atoms such that they are equivalent.

Modern ATPs generally implement some form of **indexing**, a computationally efficient method for storing and finding a particular term pattern in the clause set. One such indexing method is the discrimination tree [5, 6], which organizes atoms into a tree data structure so that matching a term pattern to an atom is a tree traversal that scales logarithmically to the size of the problem.

2.2 Hardware Parallelism

Hardware parallelism favors regular arithmetic over pointer-heavy data structures. Carryless field arithmetic is available in Intel’s GFNI architecture [7] and can be batched on GPUs [8]. In contrast, memory bandwidth and irregular pointer chasing limit GPU kernels [9, 10]. This motivates representing atom comparison as a linear algebraic operation.

2.3 Algebra using Binary Strings

The fingerprint function defined in Section 3 and the algebraic linear system based unification proposed in Section 5 operates in the 64-bit, binary finite field, utilizing XOR as the additive operation (symbolized as $\oplus$), and carry-less multiplication reduced modulo an irreducible polynomial as the multiplicative operation (symbolized as $\boxtimes$). Elements are represented as

polynomials over $\mathbb{F}_2$, where bit i of a 64 bit word is the coefficient of x^i , reduced modulo the lowest-weight irreducible polynomial of degree 64 [11]:

$$\mathbb{F} = \text{GF}(2^{64}) = \mathbb{F}_2[x]/(p(x)), \quad p(x) = x^{64} + x^4 + x^3 + x + 1 \quad (4)$$

Despite the unfamiliar operations, this structure behaves like ordinary arithmetic: $\oplus$ and $\boxtimes$ are commutative and associative, $\boxtimes$ distributes over $\oplus$, there are additive and multiplicative identities (0 and 1), every element has an additive inverse, and every non-zero element has a multiplicative inverse. These properties make the usual algebraic manipulations valid, so the techniques of Section 5 carry over from ordinary arithmetic [12].

By contrast, the mathematics utilized in Section 4 does not treat a 64-bit word as a single number. It treats it as a column vector of 64 bits, each of which is a scalar in $\mathbb{F}_2 = \{0, 1\}$ with XOR as addition and AND as multiplication. Words are combined by ordinary matrix operations, so the usual notions of matrices, kernels, and subspaces apply. Formally, this is the vector space $\mathbb{F}_2^{64}$ rather than the field $\text{GF}(2^{64})$ of Equation 4.

2.4 Hashing Functions

Hash functions map arbitrary-length bit strings to fixed-length values. Here, h is modeled as a uniform map into $\mathbb{F}$; hashing short symbols is not a bottleneck [13, 14].

$$h : \Sigma \rightarrow \mathbb{F} \text{ (uniformly random)} \quad (5)$$

3 The Fingerprint Function

Naively, the first objective can be implemented by simply hashing each atom’s textual representation. However, hashing an atom’s contents makes its structure completely opaque. This violates the second objective as the atoms $\text{married}(\text{jane}, \text{john})$ and $\text{married}(\text{jane}, X)$ should be comparable.

Instead of hashing entire atoms, we hash each term individually, then XOR them together to form the identity of the whole atom (the same XOR-composition idea behind Zobrist hashing [15]). If we modify the hash function so that all variables yield 0, they contribute nothing to the atom’s overall identity since 0 is the identity element of XOR:

$$\begin{aligned} \text{id}(\text{married}(\text{jane}, \text{john})) &= \text{id}(\text{married}) \oplus \text{id}(\text{jane}) \oplus \text{id}(\text{john}) \\ &= h(\text{married}) \oplus h(\text{jane}) \oplus h(\text{john}) \\ \text{id}(\text{married}(\text{jane}, X)) &= \text{id}(\text{married}) \oplus \text{id}(\text{jane}) \oplus \text{id}(X) \\ &= h(\text{married}) \oplus h(\text{jane}) \oplus 0 \\ &= h(\text{married}) \oplus h(\text{jane}) \end{aligned}$$

Comparing a fully grounded atom with one containing a variable costs a single XOR:

$$\text{id}(\text{married}(\text{jane}, \text{john})) \oplus \text{id}(\text{married}(\text{jane}, X)) = h(\text{john})$$

Since XOR is its own inverse, the only remaining term is $h(\text{john})$, so john is the substituent for the variable X .

One issue with this scheme is that ordinality of the terms within the atom is not captured. Since XOR is commutative, $\text{id}(\text{married}(\text{jane}, \text{john})) = \text{id}(\text{married}(\text{john}, \text{jane}))$. To address this, we introduce a new function, the **slot function**, which encodes the term’s identity and position in the atom. A term’s slot is calculated as the product of its hash and an **ordinal**

weight (w_i).

$$\begin{aligned} \text{id}(\text{married}(\text{jane}, \text{john})) &= \text{slot}(\text{married}, 0) \oplus \text{slot}(\text{jane}, 1) \oplus \text{slot}(\text{john}, 2) \\ &= (h(\text{married}) \boxtimes w_0) \oplus (h(\text{jane}) \boxtimes w_1) \oplus (h(\text{john}) \boxtimes w_2) \end{aligned}$$

These ordinal weights are fixed constants where $h(t) \boxtimes w_n$ corresponds to the identity of the term t at position n in its containing atom. Multiplication is used for the encoding as the identity of the term can be recovered from its slot value and position by multiplying it by the inverse of the ordinal weight.

$$\begin{aligned} \text{id}(\text{married}(\text{jane}, \text{john})) \oplus \text{id}(\text{married}(\text{jane}, X)) &= \text{slot}(\text{john}, 2) \\ \text{slot}(\text{john}, 2) \boxtimes w_2^{-1} &= h(\text{john}) \end{aligned}$$

Note that calculating inverses is relatively expensive, but since the ordinal weights are predetermined, their inverses can be precomputed.

For reasons that are explained in Section 5, the ordinal weights are themselves powers of a randomly selected, fixed constant z :

$$w_i = z^{i+1}, \quad z \neq 0, \quad z^i \neq z^j \text{ for all } 1 \leq i < j \leq N + 1, \quad (6)$$

where N is the maximum arity to be indexed. The first condition guarantees each w_i has an inverse; the second guarantees distinct positions receive distinct weights.

In summary, the slot function is defined as:

$$\text{slot}(t, n) = h(t) \boxtimes w_n \quad (7)$$

3.1 Subterm Representation

The fingerprint function generalizes to atoms with nested terms (e.g. $\text{married}(\text{jane}, \text{father}(\text{son}(\text{jane})))$). Rather than a single slot position, each term in an atom is located by its **path**: the sequence of argument positions traversed from the atom's root to reach it. A path $\pi = (i_0, \dots, i_d)$ has depth d with i_0 its top-level slot position. In the atom above, the predicate term, married , is at path $\pi = (0)$, the first jane is at path $\pi = (1)$, father is at $\pi = (2, 0)$, and the second jane is at path $\pi = (2, 1, 1)$.

A term t at slot i is represented as $\text{slot}(t, i) = \text{id}(t) \boxtimes w_i$. When t is compound (e.g. a function), $\text{id}(t)$ becomes an XOR of its slots. Because $\boxtimes$ is distributive over $\oplus$, the outer weight is distributed to every term inside:

$$\begin{aligned} \text{id}(\text{father}(\text{jane})) \boxtimes w_i &= ((\text{id}(\text{father}) \boxtimes w_0) \oplus (\text{id}(\text{jane}) \boxtimes w_1)) \boxtimes w_i \\ &= (\text{id}(\text{father}) \boxtimes w_0 \boxtimes w_i) \oplus (\text{id}(\text{jane}) \boxtimes w_1 \boxtimes w_i) \end{aligned}$$

A term at path $\pi = (i_0, \dots, i_d)$ is multiplied by one ordinal weight per level of nesting, so its weight is the product $w_{i_0} \boxtimes w_{i_1} \boxtimes \dots \boxtimes w_{i_d}$. Reusing the ordinal weights $w_i = z^{i+1}$ (Eq. (6)) does not work, since $\boxtimes$ is commutative. A product collapses to $z^{\sum_{k=0}^d (i_k+1)}$, so every pair of paths with equal position sums collides (e.g. $(1, 2)$ and $(2, 1)$). Therefore, we extend Eq. (6) so that each depth k is assigned a distinct base constant z_k :

$$w(\pi) = z_0^{i_0+1} \boxtimes z_1^{i_1+1} \boxtimes \dots \boxtimes z_d^{i_d+1}, \quad z_k \neq 0, \quad w(\pi) \neq w(\pi') \text{ for all } \pi \neq \pi' \quad (8)$$

The depth constants z_k are chosen independently and uniformly at random, then must be verified to produce no collisions among $w(\pi)$, up to a fixed maximum arity and depth.

The **path function** is the product of a term's identity and the weight of its path:

$$\text{path}(s, \pi) = \text{id}(s) \boxtimes w(\pi) \quad (9)$$

Note that the previously defined slot function (Eq. (7)) is the special case $\text{slot}(s, i) = \text{path}(s, (i))$.

A full construction of an atom with a nested term:

$$\begin{aligned} \text{id}(\text{married}(\text{jane}, \text{father}(\text{son}(\text{jane})))) &= \text{path}(\text{married}, (0)) \oplus \text{path}(\text{jane}, (1)) \oplus \text{path}(\text{father}, (2, 0)) \\ &\quad \oplus \text{path}(\text{son}, (2, 1, 0)) \oplus \text{path}(\text{jane}, (2, 1, 1)) \\ &= (\text{id}(\text{married}) \boxtimes z_0) \oplus (\text{id}(\text{jane}) \boxtimes z_0^2) \oplus (\text{id}(\text{father}) \boxtimes z_0^3 z_1) \\ &\quad \oplus (\text{id}(\text{son}) \boxtimes z_0^3 z_1^2 z_2) \oplus (\text{id}(\text{jane}) \boxtimes z_0^3 z_1^2 z_2^2) \end{aligned}$$

As before, the comparison between a fully constructed atom and one with a variable is a single XOR. A variable contributes 0 at any path, so the XOR of the two atoms leaves exactly the term at the variable's path, which can be recovered by multiplying it by the precomputed inverse of its path weight.

$$\begin{aligned} \text{id}(\text{married}(\text{jane}, \text{father}(\text{son}(\text{jane})))) \oplus \text{id}(\text{married}(\text{jane}, \text{father}(\text{son}(X)))) &= \text{path}(\text{jane}, (2, 1, 1)) \\ \text{path}(\text{jane}, (2, 1, 1)) \boxtimes w((2, 1, 1))^{-1} &= \text{id}(\text{jane}) \end{aligned}$$

3.2 Function Definition

The path function defined in Eq. (9) extends to arbitrary terms by recursion, accumulating the path by its traversal. Given that the empty sequence, $()$, denotes an empty path, $w() = 1$, and $\pi \cdot i$ appends element i to sequence π , the id of an atom $t_0(t_1, \dots, t_n)$ is defined as:

$$\text{path}(t, \pi) = \begin{cases} 0 & t \in \mathcal{V} \\ h(t) \boxtimes w(\pi) & t \text{ a constant or relation symbol} \\ \bigoplus_{i=0}^n \text{path}(t_i, \pi \cdot i) & t = t_0(t_1, \dots, t_n) \end{cases} \quad \text{id}(t) = \text{path}(t, ()) \quad (10)$$

Observe that a term's path value, $\text{path}(t, \pi)$, is a monomial where the constant is the hashed identity of the term and the variables are the depth constants raised to the power of the term's ordinal positions:

$$\text{path}(t, (i_0, i_1, i_2)) = h(t) \boxtimes z_0^{i_0+1} \boxtimes z_1^{i_1+1} \boxtimes z_2^{i_2+1} = c x_0^{i_0+1} x_1^{i_1+1} x_2^{i_2+1}$$

As such, an atom's total identity is a polynomial composed of those term-path monomials. This property is used in Section 3.3 to calculate the probability of two atoms' fingerprints colliding and in Section 5 to compare atoms and extract substitution bindings.

$$\begin{aligned} \text{id}(r(q, f(v, u))) &= \text{path}(r, (0)) \oplus \text{path}(q, (1)) \oplus \text{path}(f, (2, 0)) \oplus \text{path}(v, (2, 1)) \oplus \text{path}(u, (2, 2)) \\ &= c_r x_0 + c_q x_0^2 + c_f x_0^3 x_1 + c_v x_0^3 x_1^2 + c_u x_0^3 x_1^3 \end{aligned}$$

A worked construction in a 4-bit toy field appears in Appendix A (Figure 2).

3.3 Fingerprint Collisions

Each unique term has its fingerprint stored in a hash table, providing a backward lookup between the hashed identity of a term and its true identity. A lookup miss indicates that a term does not exist in the context of the problem. Collisions are avoided by rehashing a term as many times as necessary to achieve a unique identity. The resulting symbol assignments are part of the fingerprint state and are stored with the pool. This scheme is limited to exactly the number of symbols the finite addressing space supports, 2^{64} . Additionally, since variable identities are 0, a term that hashes to 0 must also be rehashed. The chance of needing to rehash a term, given m existing terms, is $m/2^{64}$.

In addition to symbols, subterms and atoms are also represented by a hashed identity and stored in respective hash tables. While symbols can be rehashed to avoid collision, compound subterms and atoms are beholden to the fingerprinting algorithm for their hashed identity and cannot have the fingerprint algorithm reapplied. Hence, collisions are expected and a subterm or atom's true identity (e.g. the data structure which represents the component terms) is kept in a list addressed by their common hashed identity. For the purposes of unification, atoms belonging to negative and positive literals are kept in separate stacks so that an atom within a negative literal only ever queries atoms of positive literals.

The collision probability between two atoms follows the Schwartz–Zippel lemma [16, 17]:

$$\Pr[\exists Q \neq R \in \mathcal{A}. \text{id}(Q) = \text{id}(R)] \leq \binom{|\mathcal{A}|}{2} \frac{D}{2^{64}} \quad (11)$$

where $\mathcal{A}$ is the atom set and D is the total degree of the fingerprint polynomial. A term at path $(i_0, \dots, i_d)$ contributes a monomial of degree $\sum_{k=0}^d (i_k + 1)$, so $D \leq (a + 1)(d + 1)$ for a maximum arity a and maximum term depth d . For a large knowledge base like SUMO [18] with approximately 200,000 unique formulas, a set maximum relation arity of 7 and a maximum observed term depth of 8, this gives $D = 72$ and a bound of roughly $2^{-23} \approx 8 \times 10^{-8}$. This represents a strict upper bound, quadratic in $|\mathcal{A}|$. At approximately 10^8 atoms, the probability is roughly 2%. Since this algorithm only compares atoms of a common arity, the size of $\mathcal{A}$ is actually an overestimate, although in most input spaces, such as TPTP [19] or SUMO, the majority of relations are binary or ternary.

Collisions become more probable as unification adds synthetic atoms, increasing the total number of atoms. Additionally, atoms differing only in variable pattern (e.g. $p(X, Y)$ and $p(X, X)$) have identical fingerprints by construction, which is not solvable via the mechanisms of the algorithm's mathematics. Therefore, any scheme which compares fingerprints requires a **verifier** to ensure correctness by checking any suggested fingerprint matches. Naively, this verification can be a per-term walk over the underlying data structure. Further strategies for efficiently detecting collisions as a part of this verifier are discussed in later sections.

4 Blackout Matching

The fingerprint function supports the comparison of two atoms through an XOR. Section 3 illustrated that when two atoms differ such that one atom contains a variable at a path where the other contains a term, XORing their fingerprints isolates the term at that path. We generalize that property of the fingerprint into a procedure, **blackout matching**. Given an atom R and a pattern Q , it removes from $\text{id}(R)$ the terms at the variable positions of Q and tests whether the result equals $\text{id}(Q)$. This decides whether $Q\sigma = R$ for some substitution σ on the variables of Q . The query is thus an instance of the candidate. Consequently, every variable position of R is also one of Q .

The terminology follows error-correcting-code decoding, from which this operation is derived. The atom to be matched is the **query** (R), the atom tested against it is the **candidate** (Q), and their XOR, $s = \text{id}(R) \oplus \text{id}(Q)$, is the **syndrome**. A **blackout** removes a subterm rooted at path π by XORing its path value out of the atom. The removed subterm is the **basis** and what remains is the **residue**. Since a variable contributes 0, the residue is the identity of the atom with that subterm replaced by a variable. For a set of paths $\mathcal{P}$:

$$\text{blackout}(R, \mathcal{P}) = \text{id}(R) \oplus \bigoplus_{\pi \in \mathcal{P}} \text{path}(t_\pi, \pi) = \text{id}(R[\mathcal{P} \mapsto \mathcal{V}]) \quad (12)$$

where t_π is the whole subterm rooted at path π in R , and $\mathcal{P}$ is an antichain: no selected path is an ancestor of another. Thus, when t_π is compound, $\text{path}(t_\pi, \pi)$ includes its root and every descendant. For example, $\text{blackout}(\text{married}(\text{jane}, \text{john}), \{(1)\}) = \text{id}(\text{married}(X, \text{john}))$, while blacking out the nested path $(1, 1)$ of $p(f(a), b)$ removes only a and yields $p(f(X), b)$.

Let $\mathcal{D}_R$ be the argument paths of all non-variable subterms of R ; function and relation symbols themselves are not argument paths. The valid blackouts use antichains of these paths. Their identities lie in the **blackout space** (B_R), obtained by XORing the $\mathbb{F}_2$ -span V_R of rooted-subterm path values against the original identity of R :

$$V_R = \text{span}_{\mathbb{F}_2} \{\text{path}(R|_\pi, \pi) : \pi \in \mathcal{D}_R\} \quad B_R = \text{id}(R) \oplus V_R \quad (13)$$

Q matches R only if $\text{id}(Q) \in B_R$, which is equivalent to testing if the syndrome $s := \text{id}(R) \oplus \text{id}(Q)$ is a member of V_R . The rooted-subterm path values of V_R form the atom's **basis set**, which can be precomputed once per query atom. Membership is a fast prefilter; the candidate's actual, structurally compatible variable roots are checked after the fold.

Testing membership by comparing s against all combinations of V_R is sound but scales poorly. Instead, we apply Gaussian elimination to the basis set to produce a **reduced basis set**: $V_R = \text{span}_{\mathbb{F}_2} \{b_1, \dots, b_r\}$ with $r \leq 64$, where each b_k has a distinct pivot bit p_k such that $(b_k)_{p_k} = 1$ and $(b_\ell)_{p_k} = 0$ for $\ell \neq k$. This is the standard-form construction familiar from parity-check matrices in coding theory [20]. The membership of a syndrome in V_R is determined by XORing each b_k into s if and only if bit p_k of s is set. If this fold reaches 0, $s \in V_R$. This is determined at a cost of at most r conditional XORs per candidate.

Additionally, during Gaussian elimination, the basis set can be augmented with an identity, so every row operation applied to the path values is applied to the identity as well. This augmented block, the **provenance matrix** (T), records for each reduced vector b_k which original path values it is a sum of [21]. It is indicative of which basis eliminations produced the residue which could unify with the given pattern. This is used in conjunction with candidate's variable roots to supply the acceptance witness.

Algorithm 1 displays the entire blackout matching algorithm. Appendix A works this algorithm end to end in the toy field (Figures 3 and 4).

4.1 Error Catching and Edge Cases

False positives are mathematically unavoidable in this scheme, as an infinite number of atomic representations are mapped to a finite field. Prevention means lowering the probability of a collision. To do so, we introduce a second fingerprint for every atom, computed with an independently drawn set of depth constants z'_k , and require that it also match before acceptance. Further fingerprints can be added; since the constants are independent, the probability of a false positive decreases exponentially for a linear increase in storage and comparison cost. Since the fingerprints are independent, the per-pair term of Eq. (11) is raised to the J th power:

$$\Pr[\exists Q \neq R \in \mathcal{A} : \text{all } J \text{ fingerprints collide}] \leq \binom{|\mathcal{A}|}{2} \left(\frac{D}{2^{64}} \right)^J \quad (14)$$

Algorithm 1 Blackout matching of query R against a candidate set $\mathcal{C}$

```

1:  $\{v_\pi\} \leftarrow \{\text{path}(R|_\pi, \pi) : \pi \in \mathcal{D}_R\}$   $\triangleright$  rooted-subterm basis set of  $R$ 
2:  $(\{b_k\}, \{p_k\}, T) \leftarrow \text{GAUSSELMINATE}(\{v_\pi\} \mid I_{|\mathcal{D}_R|})$   $\triangleright$  reduced basis, pivots, provenance
3: for all  $Q \in \mathcal{C}$  do
4:    $s_0 \leftarrow \text{id}(R) \oplus \text{id}(Q)$ ;  $s \leftarrow s_0$ 
5:   for all  $k = 1 \dots r$  do
6:     if bit  $p_k$  of  $s$  is set then  $s \leftarrow s \oplus b_k$ 
7:     end if
8:   end for
9:    $M \leftarrow \text{varroots}(Q) \cap \mathcal{D}_R$ 
10:   $d \leftarrow \bigoplus_{\pi \in M} v_\pi$ 
11:  if  $s = 0$  and  $s_0 = d$  and  $\text{varroots}(Q)$  is structurally compatible with  $R$  then report
    candidate with  $\sigma = \{X_\pi \mapsto R|_\pi : \pi \in M\}$ 
12:  end if
13: end for

```

Beyond probability, false positives are detected by auditing the candidate's actual variable roots. Let $\text{varroots}(Q)$ mark variable positions having no variable ancestor; these paths must form an antichain, since a variable replacing a subterm removes every descendant path, and must be structurally compatible with R . Compatibility requires each candidate root either to be in $\mathcal{D}_R$ or to align with a variable of R , and every variable root of R to align with a candidate variable. After membership, the candidate's expected syndrome is recomputed as $d = \bigoplus_{\pi \in M} \text{path}(R|_\pi, \pi)$, where $M = \text{varroots}(Q) \cap \mathcal{D}_R$. A legitimate match has $s_0 = d$. Positions where both atoms hold variables contribute nothing and need no blackout.

One edge case remains: a candidate with repeated variables (e.g. $Q = \text{likes}(A, A)$ against $R = \text{likes}(x, y)$). Both positions are blacked out and the fold accepts, yet Q generalizes R only if $x = y$. The candidate therefore carries a **variable group** G recording each repeated variable's positions. These groups are handed to the verifier with the candidate bindings. This is necessary for arbitrary nested occurrences, whose depth-relative encodings need not share one position-independent coefficient.

4.2 Blackout Matching as a Linear Map

The fold of Algorithm 1 makes r decisions “if bit p_k of s is set, XOR in b_k .” Because the pivots of the reduced basis set are distinct, no decision changes a bit that another decision reads. Therefore, each “decision” can be written as branchless logic: multiply the bit at p_k (1 or 0) by b_k and XOR it into s . A fixed sequence of such steps is a linear function of s , and a linear function on 64 bits is a 64×64 bit matrix:

$$\Pi_R = I + \sum_{k=1}^r b_k e_{p_k}^\top, \quad s \in V_R \iff \Pi_R s = \mathbf{0}, \quad (15)$$

where I is the standard 64×64 identity matrix and $e_{p_k}^\top$ reads bit p_k of its argument. Over $\mathbb{F}_2$, vector addition is bitwise XOR, so $+$ here is the $\oplus$ of the preceding sections, and $M + M = \mathbf{0}$. If $s \in V_R$, then $s = \sum_k c_k b_k$ (bit p_k of s is c_k), so $\Pi_R s = s + \sum_k c_k b_k = \mathbf{0}$. Conversely, $\Pi_R s = \mathbf{0}$ gives $s = \sum_k s_{p_k} b_k \in V_R$. Π_R is built once per query from the reduced basis set, and a pool of candidates is tested in a single product by placing their syndromes as the columns of a matrix S . Matches are the zero columns of $\Pi_R S$.

The membership check is linear; the candidate-specific root audit uses its stored variable-root mask after a candidate survives that check. Candidates sharing a variable-root shape can

be audited in a batch against the same precomputed expected blackout. For the arities that occur in practice the masked fold algorithm described in Algorithm 1 is cheaper and is what the experiments use. The matrix form pays off when the checks are fused or on hardware where a fixed bit-matrix apply is a primitive (four-Russians tables [22], GFNI [7], GPUs).

The stacked system is given in Appendix A (Figure 5).

5 Power Unification

Blackout matching specifically requires that the atom being queried is the instance R , which is used to search for the patterns Q that can generalize it. If the query were to be the more general atom, nothing contained within the query could be removed or added to equal a less general candidate without inspecting the structure of each candidate. For a pattern like $married(A, B)$, searching for the instances it generalizes, like $married(x, y)$, the problem becomes extracting the correct values from each candidate to match the query. In such a case, the roles of Section 4 are reversed, the pattern Q is the query and the instances R are the candidates:

$$s = \text{id}(\text{married}(A, B)) \oplus \text{id}(\text{married}(x, y)) = \text{path}(x, (1)) \oplus \text{path}(y, (2)).$$

Here, the syndrome contains information for two terms and their path. Without inspecting the contents of the candidate, one would not have the identity of either symbol, thus any solution would depend on solving for two unknown u_1, u_2 .

$$w((1)) \boxtimes u_1 \oplus w((2)) \boxtimes u_2 = s$$

Without two equations, the solution is under-determined and not solvable. However, Section 4.1 already defined additional fingerprints to reduce the probability of a collision between atom fingerprints. These separate fingerprints can be used to form and solve a system of linear equations. To create this linear system, let $c(t)$ be the fixed, depth-relative inner fingerprint of a top-level argument: $\text{path}(t, (i)) = c(t) \boxtimes z_0^{i+1}$. Thus, $c(t)$ contains every inner depth constant but no top-level weight. For a compound argument, it is not generally the global $\text{id}(t)$ illustrated in Eq. (9). We define a variation of Eq. (10) in which fingerprint j raises only the top-level weight to the j -th power:

$$\text{id}_j(P(t_1, \dots, t_n)) = h(P) \boxtimes z_0^j \oplus \bigoplus_{i=1}^n c(t_i) \boxtimes z_0^{(i+1)j} \quad (16)$$

So, fingerprint 1 uses the fingerprint function of Section 3 unchanged, fingerprint 2 squares only its top-level weights, etc. A compound binding therefore has the same value $c(t)$ in every row: inner weights are fixed across fingerprints. An earlier experimental version powered every depth constant; however, it made a compound binding recipe-dependent, so every added fingerprint introduced a new unknown as well as a new equation and could reject a true match. Solving a linear system requires that the equations of the system be linearly independent. This independence follows from the Vandermonde argument in Section 5.1. This method of solving a system of linear equations to recover the original values of a syndrome is an already established technique used in erasure decoding and BCH codes [23, 24, 20] ($S_j = \sum_i u_i x_i^j$ with known locations x_i and unknown values u_i).

The collision-control fingerprints of Section 4.1 use independently drawn sets of depth constants, satisfying the independence requirement of the Schwartz–Zippel lemma and Eq. (14). The power fingerprints above are instead a correlated family derived by varying one z_0 . If the same fingerprints are reused for both purposes, the exponential improvement of Eq. (14)

does not apply. Requiring all of them to collide is still bounded by the first-fingerprint term $\binom{|A|}{2} D/2^{64}$, and a sharper J -dependent bound would require a separate analysis.

Given two atoms $Q = P(Q_1, \dots, Q_n)$ and $R = P(R_1, \dots, R_n)$, let U be the set of top-level argument positions at which either argument is non-ground. For a true match, positions outside U agree and cancel. The $k = |U|$ unknowns are $u_\ell = c(Q_{i_\ell}) \oplus c(R_{i_\ell})$, where $i_\ell \in U$. Each atom carries at least k fingerprints. If Q_{i_ℓ} is a variable, $u_\ell = c(R_{i_\ell})$ identifies its entire binding, including a compound term. If Q_{i_ℓ} contains nested variables, $c(R_{i_\ell}) = u_\ell \oplus c(Q_{i_\ell})$ identifies the facing top-level term and the verifier splits the nested bindings. If both positions hold variables, $u_\ell = 0$ and the verifier receives the variable-to-variable link. For $Q = p(A, x)$ and $R = p(y, B)$, $U = \{1, 2\}$, $A \mapsto u_1$ and $B \mapsto u_2$. Solving the system below returns $u_1 = h(y)$, $u_2 = h(x)$, giving $\sigma = \{A \mapsto y, B \mapsto x\}$. Therefore, take the differences of each atom’s corresponding fingerprints, $S_j = \text{id}_j(Q) \oplus \text{id}_j(R)$, and solve for the u_ℓ in the linear system

$$S_j = \bigoplus_{\ell=1}^k u_\ell \boxtimes z_0^{(i_\ell+1)j}, \quad 1 \leq j \leq k, \quad (17)$$

where i_ℓ is the ℓ -th top-level position in U .

5.1 Power Unification as Linear Algebra

We describe how extracting bindings between variables and terms can be accomplished through solving a system of linear equations. Now, we describe how this system can be expressed as a linear algebraic operation and support multiple comparisons simultaneously.

A pair of atoms with k unknowns at top-level positions $i_1, \dots, i_k$ generates k equations in the k unknowns $u_1, \dots, u_k$. In other words, to solve for k unknowns you need k fingerprints. Treating the top-level weights as points in the system, $x_\ell = z_0^{i_\ell+1}$, the system organizes into a matrix where every equation becomes a row and every unknown a column,

$$\mathbf{W}\mathbf{u} = \mathbf{S}, \quad \mathbf{W}_{j,\ell} = x_\ell^j, \quad (18)$$

where $\mathbf{W}$ holds the coefficients (the top-level weights of the unknown positions), $\mathbf{u}$ the bindings to extract, and $\mathbf{S}$ the fingerprint differences. This follows from Eq. (17) by interpreting the summation over ℓ as a matrix-vector product, where each element S_j is computed as the dot product of the j -th row of $\mathbf{W}$ and the vector $\mathbf{u}$.

To trust the bindings of this system means that whenever a true match exists, the system has exactly one solution. This follows from the determinant. $\mathbf{W}$ is a Vandermonde matrix [21], with

$$\det \mathbf{W} = \left(\prod_{\ell} x_\ell \right) \boxtimes \prod_{\ell < m} (x_m \oplus x_\ell),$$

and every factor is nonzero because the points are pairwise distinct and nonzero and the field has no zero divisors. The points are distinct powers of z_0 , so the distinctness condition of Eq. (6) suffices. Hence $\det \mathbf{W} \neq 0$, and if $\mathbf{u}$ and $\mathbf{u}'$ both solve the system then $\mathbf{W}(\mathbf{u} \oplus \mathbf{u}') = \mathbf{0}$ forces $\mathbf{u} = \mathbf{u}'$. In other words, solving the system finds the bindings for substitution.

Since $\det \mathbf{W} \neq 0$, solving reduces to

$$\mathbf{u} = \mathbf{W}^{-1}\mathbf{S}. \quad (19)$$

Since the points of the linear system come only from the unknown positions, $\mathbf{W}^{-1}$ is built once per shape by standard matrix inversion, ahead of any candidate or operation, and stored. Candidates are grouped by compatible unknown-position shape, and each group shares one $\mathbf{W}^{-1}$.

Within one such group, XOR each candidate’s fingerprints against the pattern’s to build one syndrome column. Stacking candidates side by side turns the group into a syndrome matrix $\mathbf{S}$, and the columns never interact. One matrix multiplication between the syndrome matrix and the inverse weights matrix then decodes every candidate in that group at once. The product matrix’s row ℓ holds every candidate’s top-level difference u_ℓ . Recover $c(R_{i_\ell}) = u_\ell \oplus c(Q_{i_\ell})$ and check it against the term lookup table. If Q_{i_ℓ} is a variable, this directly recovers its binding; nested variables sharing a top-level argument are split by the verifier. Any difference outside U , whether a different predicate or a differing ground term, is folded into the syndrome and corrupts every recovered value, so such candidates fail the lookup except with the collision probability of Section 3.3. The verifier also enforces repeated-variable consistency and the occurs check, which the arithmetic does not see. The true match arrives with its top-level substitution already assembled.

Figure 6 in Appendix A shows this product worked in the toy field.

5.2 Storage and Precomputation

A pair of atoms with k unknowns requires k fingerprints on each atom, and k is bounded by the arity, so each atom gets fingerprints up to its arity by default. Two fingerprints were already stored for collision avoidance (subject to the reuse caveat above), and most relations are binary, at least in SUMO and TPTP, though larger atoms need more fingerprints. The main overhead is the inverse weight matrix, but it comes from a closed, finite set of shapes that can be precomputed. A shape is the set of top-level unknown positions together with its partition into repeated variables; nested repetition is handed to the verifier. Shapes are counted by the Bell numbers [25], $\text{\#shapes}(n) = \sum_{r=1}^n \binom{n}{r} B_r$, so atoms with arity 1 require 1 shape, arity 2 requires 4, arity 3 requires 14, and arity 7 requires 4139.

5.3 Relation to Blackout Matching

Both procedures apply one precomputed matrix to a syndrome matrix. Blackout matching applies Π_R , 64×64 over $\mathbb{F}_2$, and tests membership so that the matches are represented as the zero columns of $\Pi_R S$. Power unification applies $\mathbf{W}^{-1}$, $k \times k$ over $\text{GF}(2^{64})$, and recovers the bindings as the entries of $\mathbf{W}^{-1} \mathbf{S}$. In coding vocabulary, Π_R is the detector and $\mathbf{W}^{-1}$ the decoder.

6 Experimentation

6.1 The Corpus

We used a classified version of a portion of the SUMO knowledge base as the test set as it provided a singular, standard format collection of axioms and atoms to test against. It contained 115,737 formulas, 26,954 distinct symbols, and 72.4% flat atoms with no nested argument. All experiments were run over the 52,878 atoms distinct up to renaming, almost exclusively at arity 2 and 3. The top two predicates (`instance` and `subclass`) account for 45% of all atoms. Additionally, about 21% of atoms are some variant of `(instance ?X <something>)`.

6.2 Fingerprint Collisions

Although avoiding collisions in a fingerprinting algorithm that maps to a finite space is impossible, a uniform 64-bit hash collides with probability $\frac{1}{2^{64}}$ for a fixed pair. Schwartz-Zippel gives

the corpus-wide bound in Eq. (11). We next empirically illustrate these rates against the test corpus.

The first test measured the collision rate for bare symbolic constants interned into a common hash table with hash bit width swept from 16-bits to 64-bits. The results track the birthday bound at every hash width, and *symbol collisions reached zero from 40 bits upward*.

The frequency of atom fingerprint collision was measured next. The fingerprint of each atom was calculated for a random weight-constant seed, z , and at increasing bit lengths. The total collisions between any two atoms' fingerprinted identity was divided by the collision count that a pseudo-random hash function of the same atom at the same width would produce (the pseudo-random baseline). The result of 1 is indicative that the scheme has the same pseudorandom output distribution that a hash function at the same bit width has. The resulting ratios stayed between 0.85 and 1.28 at every width tested. Collisions reach zero at 32 bits.

6.3 Storage Complexity

Minimizing memory requirements similarly minimizes the frequency and cost of memory boundary movements (e.g. moving from RAM to CPU cache) of unification chunks. Therefore, real measurements of the algorithm's storage footprint were measured as a tradeoff to increasing the number of fingerprints stored for each atom (J).

The largest static storage requirement of the scheme is the precomputed power unification inverse weight matrices. One matrix is required for each possible top-level atom shape. An atom's shape is the set of its top-level positions containing a variable, together with any repeated variables across those positions. Nested variables are handled by the verifier and do not create further matrix shapes.

The second test measured the scaling of the index's space complexity for increasing values of J over the test corpus. The test observed that, for $J = 1$, the implementation required 1.60 machine words per atom. This number increased linearly to 4.60 machine words per atom for $J = 4$ and 8.60 at $J = 8$. This relationship, linear scaling with a constant overhead is consistent with the theoretical expectation.

6.4 Correctness

While increasing the number of fingerprints, J , increases the necessary memory for each atom, it was also theorized to reduce the rate at which atoms are mismatched. Three implementations of the indexing algorithm were instrumented: **detect** used only the first fingerprint to decide whether two atoms were unifiable and did not attempt to extract substitution bindings, **full** used the joint stratified solve over both atoms' unknown positions, and **directed** first attempted directional solves and then fell back to the joint solve.

Table 1 describes the correctness experiment over varying values for J and varying algorithm strategies. The false positive (FP) rate describes the portion of pairs that were not unifiable that were identified as unifiable. Reduction is the number of tested pairs divided by the number of proposals. Determined counts all proposed candidates with a full-column-rank algebraic recovery, including false positives. The denominator of $\hat{\sigma}$ counts only determined true positives. The **detect** rows are invariant in J by construction because they use only the first fingerprint. False positives fall through $J = 4$, while false negatives remain zero in **full** and **directed** mode. These measurements evaluate only the mathematical prefilter. They do not apply the supporting verifier, including term-table lookup, repeated-variable checks, the occurs check, or a syntactic term walk. Thus, the reported false-positive rates are raw prefilter proposal rates rather than final audited false-positive rates.

Table 1: Power-unification correctness summary: 14,529,000 unification pairs tested, of which 27,731 were unifiable. FP rate over non-unifiable pairs; FN rate over unifiable pairs; $\hat{\sigma}$ is the conditional sigma-exact rate with its denominator (determined true positives).

J	mode	FP rate	FN rate	reduction	determined	$\hat{\sigma}$ (num/den)
1	detect	0.8793	0.0000	1.1×	2,771,398	1.000 (37/37)
	full	0.8793	0.0000	1.1×	2,771,398	1.000 (37/37)
	directed	0.8793	0.0000	1.1×	5,895,265	0.002 (46/23103)
2	detect	0.8793	0.0000	1.1×	2,771,398	1.000 (37/37)
	full	0.6372	0.0000	1.6×	5,095,990	1.000 (1906/1906)
	directed	0.6372	0.0000	1.6×	9,024,550	0.092 (2535/27634)
4	detect	0.8793	0.0000	1.1×	2,771,398	1.000 (37/37)
	full	0.0171	0.0000	52.6×	90,790	1.000 (1938/1938)
	directed	0.0171	0.0000	52.6×	276,295	0.095 (2636/27730)
8	detect	0.8793	0.0000	1.1×	2,771,398	1.000 (37/37)
	full	0.0171	0.0000	52.6×	90,790	1.000 (1938/1938)
	directed	0.0171	0.0000	52.6×	276,295	0.095 (2636/27730)

To test completeness directly, we first used 617 syntactically unifiable pairs and split them by their known substitutions: 599 pairs bound every variable to a constant, while 18 bound at least one variable to a compound term. This direct test is separate from the sampled sweep of Table 1. The scheme produced zero false negatives at every J in both modes. It recovered the substitution exactly for 598 constant-binding pairs and 7 compound-binding pairs; it reported the remaining 1 and 11 cases as partial, meaning that the arithmetic identified the facing top-level term but left nested bindings for exact syntactic verification.

We also used syntactic unification as the reference while sweeping every same-arity candidate for 400 sampled queries. We chose 300 2-arity atoms to compare against 46,274 other 2-arity atoms, and 100 3-arity atoms to compare each against 6,468 other 3-arity atoms. Thus, $300 \cdot 46,274 + 100 \cdot 6,468 = 14,529,000$ pairs were tested, with zero false negatives in **full** and **directed** mode through $J = 8$. The raw prefilter false-positive rate fell to 0.017 at $J = 4$ and remained at that rate through $J = 8$. We do not interpret this residual as a 64-bit collision estimate because the power fingerprints are correlated. On a separate stratified random sample of 80 queries and 400 candidates per segment at $J = 4$, the raw prefilter achieved 52.2× candidate reduction.

6.5 Parallel Performance Scaling

The core claim of this work is that the algorithm specified here allows for consistent performance scaling with parallelized hardware. The performance experiment repeated the correctness test, added wall-clock timing, and parallelized unification matching across multiple CPU cores and on a 40-core GPU. Figure 1 illustrates the linear relationship between added cores and performance, as well as the further reduction in wall-clock time for the GPU implementation. This relationship is indicative of a lack of cross-thread dependency and blocking. On this workload, however, the packed-matrix prefilter does not yet outperform two-worker Robinson unification [4] in absolute time. The $J = 2$ GPU measurement is 0.64s, compared with 0.32s for Robinson. The figure therefore demonstrates parallel scaling rather than an end-to-end speedup.

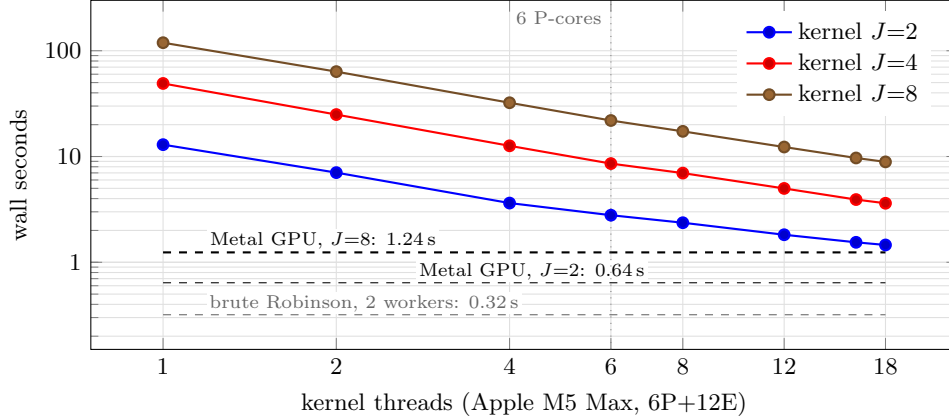


Figure 1: Wall-clock time for the generic packed-matrix unification kernel over varying degrees of parallelism. Each measurement covers 50 sweeps over 46,275 candidates and reports the best of three runs. These are prefilter kernel times and exclude syntactic verification.

7 Future Work

7.1 Implementation

This fingerprinting, indexing, and unification scheme is implemented in the core saturation loop of the SUMO Prover (SUPr), an ATP designed specifically to reason against SUMO. SUPr competed in the 2026 iteration of the IJCAR ATP System Competition (CASC-J13) [26]. The scheme arose in part from the need to cache and rehydrate an axiom index between runs without rebuilding a pointer-based structure. Its fingerprint state consists of flat arrays, the symbol table, and precomputed shape tables. The system maintains soundness by a term-by-term verifier for proposed unifications. The CASC-J13 version used SIMD only for flat atoms and fell back to a linear search for nested terms. Future versions will implement the full scheme and evaluate GPU execution.

7.2 Obfuscated Theorem Proving

If h were a cryptographically secure keyed hash [13] and its inputs were salted with a secret held only by trusted endpoints, a remote prover could compare opaque fingerprints without seeing their plaintext symbols. This is not a confidentiality mechanism. Salted fingerprinting is deterministic encryption, which leaks equality patterns [27] and is vulnerable to frequency analysis against public auxiliary data [28, 29]. It provides obfuscation comparable to a symbol-stripped binary.

7.3 Speculative Saturation

The algorithms presented here are approximate, so a practical implementation could deliberately generate speculative inferences but bound them with a strict auditor. This inverts the conventional architecture, where symbolic checks occupy the hottest part of the saturation loop. A GPU could instead produce candidate inferences as fingerprint algebra while a trusted auditor syntactically replays only the derivations relevant to a final proof. Such a system is exactly as sound as its auditor, as in certificate-checking SAT [30, 31] and proof-reconstruction

workflows [32]. If an audit reveals a collision, rekeying the hash salt or generator z relocates it without invalidating previously verified derivations.

Acknowledgements

The authors would like to thank Stephan Schulz and Pante Stanica for their helpful comments on a draft of this paper. Additionally, the authors are thankful for the thorough review this paper received as a part of its submission to the International Workshop on the Implementation of Logic (IWIL 2026).

References

- [1] Stephan Schulz. Fingerprint indexing for paramodulation and rewriting. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning – 6th International Joint Conference, IJCAR 2012*, volume 7364 of *Lecture Notes in Computer Science*, pages 477–483, Berlin, Heidelberg, 2012. Springer.
- [2] William McCune. Otter 3.3 reference manual. Technical Report ANL/MS-C-TM-263, Argonne National Laboratory, 2003.
- [3] Stephan Schulz. E – a brainiac theorem prover. *AI Communications*, 15(2-3):111–126, 2002.
- [4] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
- [5] William McCune. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9:147–167, 1992.
- [6] R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov. Term indexing. In Alan J. A. Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 26, pages 1853–1964. Elsevier and MIT Press, 2001.
- [7] Intel Corporation. Galois field new instructions (GFNI) technology guide. Technical report, Intel Corporation, 2021.
- [8] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, 2008.
- [9] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer. AI and memory wall. *IEEE Micro*, 44(3):33–39, 2024.
- [10] Maria Paola Bonacina. Parallel theorem proving. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, chapter 6. Springer, 2018.
- [11] Gadiel Seroussi. *Table of low-weight binary irreducible polynomials*. Hewlett-Packard Laboratories Palo Alto, California, 1998.
- [12] Rudolf Lidl and Harald Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2nd edition, 1997.
- [13] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press (Chapman & Hall/CRC Cryptography and Network Security Series), 3rd edition, 2020. See Appendix A.4 “The Birthday Problem” and §4.6.3 “A Generic ‘Birthday’ Attack”.
- [14] Yann Collet. xxHash: Extremely fast hash algorithm. <https://github.com/Cyan4973/xxHash>, 2024. GitHub repository, includes XXH64 and XXH3.
- [15] Albert L. Zobrist. A new hashing method with application for game playing. Technical Report TR#88, Computer Sciences Department, University of Wisconsin–Madison, 1970. Reprinted in ICCA Journal 13(2):69–73, 1990.
- [16] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.
- [17] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation (EUROSAM ’79)*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226, Berlin, Heidelberg, 1979. Springer.
- [18] Ian Niles and Adam Pease. Towards a standard upper ontology. In Chris Welty and Barry Smith, editors, *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, pages 2–9, 2001.
- [19] Geoff Sutcliffe. The TPTP problem library and associated infrastructure: From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.
- [20] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*, volume 16 of *North-Holland Mathematical Library*. North-Holland, Amsterdam, 1977.
- [21] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2nd edition, 2013.
- [22] Martin R. Albrecht, Gregory V. Bard, and William Hart. Algorithm 898: Efficient multiplication of dense matrices over GF(2). *ACM Transactions on Mathematical Software*, 37(1):1–14, 2010.
- [23] Alexis Hocquenghem. Codes correcteurs d’erreurs. *Chiffres*, 2:147–156, 1959.
- [24] R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.
- [25] Gian-Carlo Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5):498–504, 1964.
- [26] Geoff Sutcliffe, editor. *Proceedings of the 13th IJCAR ATP System Competition (CASC-J13)*, Lisbon, Portugal, July 2026. Held at the 13th International Joint Conference on Automated Reasoning (IJCAR 2026) and the Federated Logic Conference (FLoC 2026).

- [27] Mihir Bellare, Alexandra Boldyreva, and Adam O’Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *LNCS*, pages 535–552. Springer, 2007.
- [28] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 644–655, 2015.
- [29] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, pages 85–100, 2011.
- [30] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing (SAT 2014)*, volume 8561 of *LNCS*, pages 422–429. Springer, 2014.
- [31] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In *Automated Deduction – CADE 26*, volume 10395 of *LNCS*, pages 220–236. Springer, 2017.
- [32] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. *Journal of Automated Reasoning*, 51(1):109–128, 2013.

A Worked Examples in the Toy Field

The following figures demonstrate example applications of the fingerprinting, blackout matching, and power unification algorithms worked in the 4-bit field $\text{GF}(2^4)$ with modulus $x^4 + x^3 + 1$. We use $z = 0010$, so the ordinal weights are $z^1 = 0010$, $z^2 = 0100$, and $z^3 = 1000$.

A.1 Fingerprint Construction

Figure 2 gives the toy symbol hashes and ordinal weights, the fingerprints they produce for a small atom pool, and one expansion worked in full. Variables (A, B) hash to 0 and contribute nothing.

<u>Symbols</u>	<u>Atoms</u>	<u>Worked expansion</u>
$\text{id}(\text{married}) = 0110$	$\text{id}(\text{married}(A, B)) = 1100$	$\text{slot}(\text{likes}, 0,) = 0010 \boxtimes 0010$
$\text{id}(\text{likes}) = 0010$	$\text{id}(\text{likes}(x, c)) = 0001$	$= 0100$
$\text{id}(x) = 0100$	$\text{id}(\text{married}(x, b)) = 0010$	$\text{slot}(x, 1,) = 0100 \boxtimes 0100$
$\text{id}(b) = 1001$	$\text{id}(\text{likes}(A, B)) = 0100$	$= 1001$
$\text{id}(c) = 1101$	$\text{id}(\text{likes}(A, c)) = 1000$	$\text{slot}(c, 2,) = 1101 \boxtimes 1000$
	$\text{id}(\text{likes}(A, b)) = 0011$	$= 1100$
<u>Weights</u>		$\text{id}(\text{likes}(x, c)) = 0100 \oplus 1001 \oplus 1100$
$w = \{0010, 0100, 1000\}$		$= 0001$

Figure 2: Fingerprint construction in the toy field: symbol hashes and ordinal weights (left), the resulting atom fingerprints (center), and the expansion of $\text{id}(\text{likes}(x, c))$ (right).

A.2 Blackout Matching

Figure 3 collapses the Gaussian elimination of the basis set for the query $\text{likes}(x, c)$: the basis vectors $V_{R(\text{likes}, x, c)} = \{0000, 1001, 1100, 0101\}$ are reduced by XORing each pivot row downward to Row Echelon Form, then upward, yielding the reduced basis. Figure 4 then collapses the SIMD application of that reduced basis against the remaining atom pool: the register holds each candidate’s syndrome against the query, each reduced basis vector is folded in wherever its pivot bit is set, and the zero rows are the reported matches. Finally, Figure 5 demonstrates the same blackout matching procedure expressed as a linear algebraic transformation.

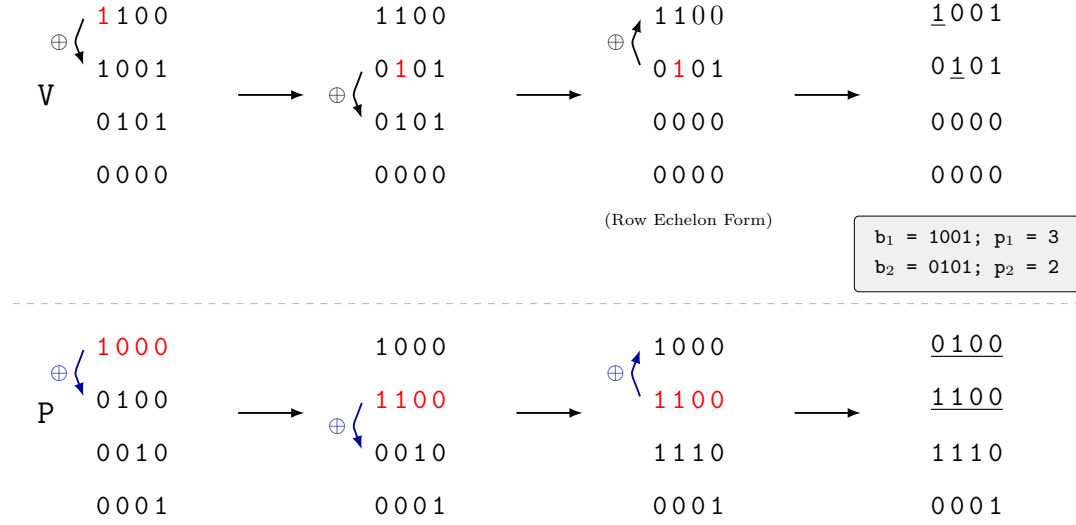


Figure 3: Gaussian elimination of the basis set for $likes(x, c)$. Red bold digits mark the active pivot, curved arrows the row being XORed. The downward pass reaches Row Echelon Form, the upward pass removes the 1 above the second pivot, and the surviving rows form the reduced basis b_1, b_2 with exclusive pivot bits p_1, p_2 . The bottom portion shows the creation of the provenance matrices. This example demonstrates that the first reduced basis element is derived from the first blackout and the second is from the next two.

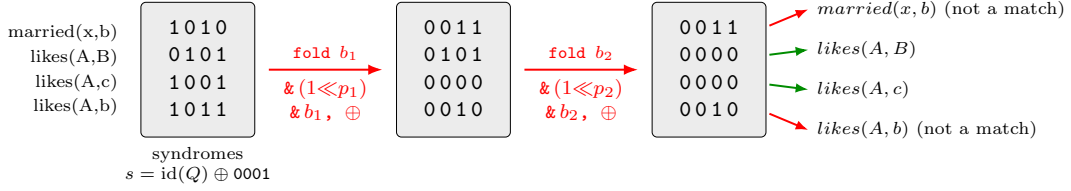


Figure 4: SIMD application of the reduced basis against the candidate pool, collapsed from the presentation's animation. The register holds each candidate's syndrome against the query $likes(x, c) = 0001$. Each reduced basis vector is folded in wherever its pivot bit is set (mask by $1 \ll p_k$, smear, AND with b_k , vector XOR). Zero rows are reported as matches.

A.3 Power Unification as One Matrix Product

Figure 6 shows the whole-pool solve for the pattern $likes(A, B)$ under the power weight scheme. It uses the same toy symbol table and powers $z^1 = 0010$, $z^2 = 0100$, $z^3 = 1000$, $z^4 = 1001$, $z^5 = 1011$, and $z^6 = 1111$. Each column of $\mathbf{S}$ is one candidate's syndrome pair $S_j = id_j(\text{candidate}) \oplus id_j(\text{pattern})$, for the candidates $likes(x, c)$, $married(x, b)$, $likes(A, c)$, $likes(A, b)$, $married(A, B)$ in column order.

P: pivot-reading rows

$$P = \begin{bmatrix} e_{p_1}^\top \\ e_{p_2}^\top \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$p_1 = 3, p_2 = 2$ (from Fig. 3)

B: reduced basis as columns

$$B = [b_1 \ b_2] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$b_1 = 1001, b_2 = 0101$ (from Fig. 3)

$$\overbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}}^B \overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}^P = \overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}}^{BP} \oplus \overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^I = \overbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}}^{\Pi_R}$$

Rows p_1, p_2 (shaded) vanish identically: reduced echelon form guarantees each pivot column has a lone 1 in its own row, so BP reproduces I exactly at every pivot row.

$$\Pi_R s = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \neq \mathbf{0}$$

$s = 1011$, the syndrome of $\text{likes}(A, b)$ against $\text{likes}(x, c)$: a nonzero residue rejects it, exactly matching the SIMD result in Figure 4.

Figure 5: Assembling the reduced basis from Figure 3 into the blackout projector $\Pi_R = I \oplus BP$ (15), and applying it to one candidate syndrome. P reads the pivot bits, B reconstructs the span member they describe, and XORing that off against I leaves the residue. The single matrix-vector product reproduces exactly what the fold pipeline in Figure 4 computes one candidate at a time; stacking candidate syndromes as columns of Σ gives the batch form $\Pi_R \Sigma$.

$$\overbrace{\begin{bmatrix} 0100 & 1100 \\ 0001 & 0110 \end{bmatrix}}^{W^{-1}} \times \overbrace{\begin{bmatrix} 0101 & 0110 & 1100 & 0111 & 1000 \\ 1011 & 1100 & 0100 & 1010 & 1001 \end{bmatrix}}^S = \overbrace{\begin{bmatrix} 0100 & 0111 & 0000 & 0000 & 0011 \\ 1101 & 0101 & 1101 & 1001 & 1100 \end{bmatrix}}^U$$

Figure 6: Solving the whole pool at once: $U = W^{-1}S$. Row 1 of U is every candidate's binding for A , row 2 for B . Green columns in the result matrix, U , are full matches ($\text{likes}(x, c)$: $A \mapsto 0100 = x, B \mapsto 1101 = c$), gray columns are partial matches (0000 signals a variable-to-variable link), and red columns are rejects, whose decoded values appear in no symbol table.