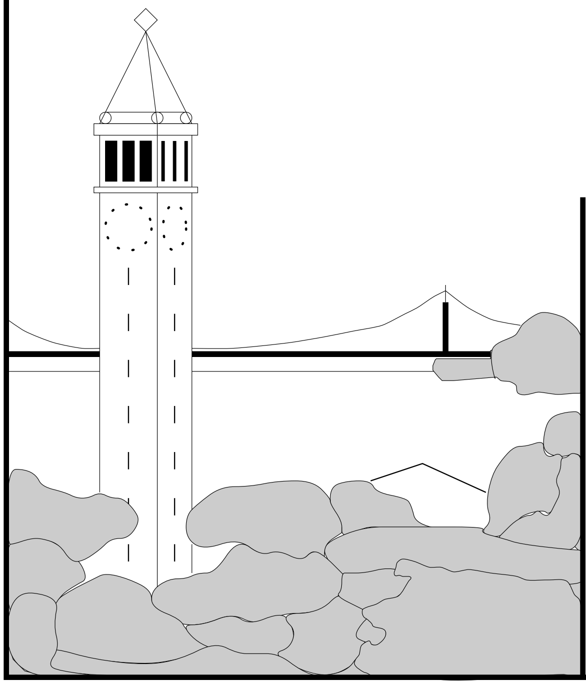
A Randomized Satisfiability Procedure for Arithmetic and Uninterpreted Function Symbols

(Full Version)

Sumit Gulwani and George C. Necula



Report No. UCB/CSD-3-1241

April 2003

Computer Science Division (EECS)
University of California
Berkeley, California 94720

A Randomized Satisfiability Procedure for Arithmetic and Uninterpreted Function Symbols
(Full Version)

Sumit Gulwani and George C. Necula

University of California, Berkeley
{gulwani,necula}@cs.berkeley.edu

Abstract. We present a new randomized algorithm for checking the satisfiability of a conjunction of literals in the combined theory of linear equalities and uninterpreted functions. The key idea of the algorithm is to process the literals incrementally and to maintain at all times a set of random variable assignments that satisfy the literals seen so far. We prove that this algorithm is complete (i.e., it identifies all unsatisfiable conjunctions) and is probabilistically sound (i.e., the probability that it fails to identify satisfiable conjunctions is very small). The algorithm has the ability to retract assumptions incrementally with almost no additional space overhead. The key advantage of the algorithm is its simplicity. We also show experimentally that the randomized algorithm has performance competitive with the existing deterministic symbolic algorithms.

1 Introduction

In this paper, we consider the problem of checking the satisfiability of a formula that involves linear equalities and uninterpreted function symbols, and explore what can be learned about the formula by evaluating it over some randomly chosen variable assignments.

Consider, for example, the following formulas ϕ_1 and ϕ_2.

$$\phi_1 : (z = x + y) \wedge (x = y) \wedge (z \neq 2x)$$

$$\phi_2 : (z = x + y) \wedge (x = y) \wedge (z \neq 0)$$

The formula ϕ_1 is unsatisfiable because no assignment that satisfies the constraint $(z = x + y) \wedge (x = y)$ also satisfies the constraint $(z \neq 2x)$. In other words, the solution space L for the constraint $(z = x+y) \wedge (x = y)$ is included in the solution space R_1 for the constraint $(z = 2x)$, as shown in Figure 1(a). On the

This research was supported in part by the National Science Foundation Career Grant No. CCR-9875171, and ITR Grants No. CCR-0085949 and No. CCR-0081588, and gifts from Microsoft Research. The information presented here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

other hand, the formula ϕ_2 is satisfiable because there exists at least one solution that satisfies the constraint $(z = x+y) \wedge (x = y)$ as well as the constraint $(z \neq 0)$. In other words, the solution space L for the constraint $(z = x + y) \wedge (x = y)$ is not included in the solution space R_2 for the constraint $z = 0$, as shown in Figure 1(b). In general, a conjunction of literals is unsatisfiable if and only if the solution space for all of the equality literals is included in the solution space for the negation of one of the disequality literals.

Can we decide the satisfiability of these formulas by evaluating them over some random values? If we choose arbitrary random values for x, y and z, then, very likely, they will not satisfy the constraint $(z = x + y) \wedge (x = y)$ (and hence they will satisfy neither ϕ_1 nor ϕ_2). Thus, such a naive "test" fails to discriminate between satisfiable and unsatisfiable formulas. However, if we manage to choose random values for x, y and z from the solution space L, then they will still not satisfy formula ϕ_1, but, very likely, they will satisfy formula ϕ_2. This is because, as shown in Figure 1(b), there is only one point P $(x = y = z = 0)$ in L that also lies in R_2, and it is extremely unlikely that when we choose a point randomly on the line represented by L, we choose the point P. In general, if a formula is unsatisfiable, then any randomly chosen assignment does not satisfy the formula. On the other hand, if a formula is satisfiable, an assignment that satisfies the equality literals in the formula, very likely also satisfies the disequality literals in the formula. We can further reduce the probability of error by choosing several random points from L rather than just one. These observations form the basis for our randomized algorithm for deciding the satisfiability of a formula.

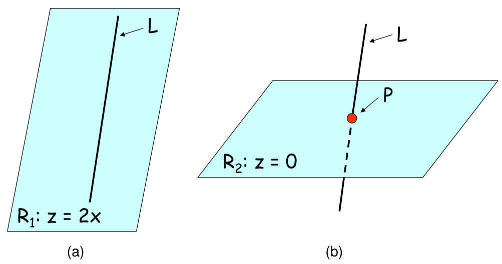


Fig. 1. The line L represents the solution space for the constraint $(z = x+y) \wedge (x = y)$. If we choose points randomly on L, we can easily tell that $L \Rightarrow R_1$ and $L \not\Rightarrow R_2$.

The key step in our algorithm is to generate random assignments that satisfy all of the equality literals. We do this incrementally, by starting with a set of completely random assignments and then adjusting them so that they satisfy each equality literal in turn. The adjustment operation can be viewed geometrically as a projection onto the hyperplane represented by an equality literal.

As we will see, this algorithm is simple and efficient. It avoids the need for symbolic manipulation and construction of normal forms. Handling arithmetic expressions becomes especially easy because we only evaluate them rather than manipulating them symbolically. Furthermore, we require a simple data structure (a set of variable assignments and a hash table for handling uninterpreted function symbols), and we perform only simple arithmetic operations.

We start with a discussion of the notation in Section 2. In Section 3, we describe the algorithm for the arithmetic fragment along with the proof of completeness, and a sketch of the proof of probabilistic soundness (the complete proof is in Appendix A) . In Section 4, we extend the algorithm to handle uninterpreted function symbols. In Section 5, we show that it is quite easy to also retract equality literals (a property that is useful in the context of a Nelson-Oppen theorem prover). In Section 6, we describe our initial experience with an implementation of this algorithm, and we compare it with a deterministic satisfiability algorithm for the same theory.

2 Notation

Consider the following language of terms over rationals $\mathbb{Q}$.

$$t ::= x \quad | \quad q \quad | \quad t_1 + t_2 \quad | \quad t_1 - t_2 \quad | \quad q \times t \quad | \quad f(t_1, .., t_k)$$

Here $q \in \mathbb{Q}$, x is some variable and f is some k-ary uninterpreted function symbol for some non-negative integer k. An *equality literal* is an equality of the form $t = 0$ while a *disequality literal* is a disequality of the form $t \neq 0$ for some term t. A formula ϕ is a set of equality and disequality literals.

An *assignment* ρ for n variables maps each variable to a rational value. We use the notation $\rho(x)$ to denote the value of variable x in assignment ρ. Occasionally, in order to expose the geometric intuition behind the algorithms, we also refer to the n variables as *coordinates* and to an assignment as a *point* in $\mathbb{Q}^n$. We write $[\![t]\!]\rho$ for the meaning of term t in the assignment ρ (using the usual interpretation of the arithmetic operations over $\mathbb{Q}$). An assignment ρ satisfies an equality $t = 0$ (written $\rho \models t = 0$) when $[\![t]\!]\rho = 0$.

We refer to a sequence of assignments S as a *sample* and we write S_i to refer to the i^{th} element of the sample S. In the geometric interpretation, a sample is a sequence of points. A sample satisfies a linear equality $t = 0$ when all of its assignments satisfy the equality. We write $S \models t = 0$ when this is the case.

An *affine combination* of two assignments ρ_1 and ρ_2 with weight $w \in \mathbb{Q}$ (denoted by $\rho_1 \oplus_w \rho_2$) is another assignment ρ such that for any variable x, $\rho(x) = w \times \rho_1(x) + (1 - w) \times \rho_2(x)$. If the assignments ρ_1 and ρ_2 are viewed as points in $\mathbb{Q}^n$ then their affine combinations are the points situated on the line passing through ρ_1 and ρ_2. The affine combination of two assignments has the property that it satisfies all the linear equalities that are satisfied by both the assignments.

3 The Algorithm for the Arithmetic Fragment

We start with a discussion of the satisfiability algorithm for formulas that do not contain any uninterpreted function symbols. We first describe the `Adjust` operation and then show how it can be used to check the satisfiability of a formula.

3.1 The `Adjust` Operation

The `Adjust` operation takes a sample S and a term e, and produces a new sample S' such that S' satisfies all the linear equalities that are satisfied by S and exactly one more linearly independent equality $e = 0$. For this definition to be meaningful, the `Adjust` operation has a precondition that $S \not\models e + c = 0$ for any constant c. Note that if this precondition does not hold and $c = 0$, then since S already satisfies $e = 0$, there is no need for the `Adjust` operation; and if $c \neq 0$, then S' cannot simultaneously satisfy $e + c = 0$ and $e = 0$. In the latter case, the formula being checked is declared unsatisfiable.

The resulting sample S' has the following properties:

A1. For any term t, if $S \models t = 0$, then $S' \models t = 0$.
A2. $S' \models e = 0$.
A3. For any term t, if $S' \models t = 0$, then $\exists \lambda$ such that $S \models t + \lambda e = 0$.

The property A1 says that the sample S' continues to satisfy all the linear equalities that are satisfied by the sample S, while the property A2 says that the sample S' also satisfies the equality $e = 0$. The property A3 implies that S' satisfies exactly one more linearly independent equality than those satisfied by S.

An Implementation of the `Adjust` Operation We now present an efficient implementation of the `Adjust` operation, assuming the precondition $\neg(\exists c \in \mathbb{Q}.\ S \models e + c = 0)$:

1 `Adjust(`$[S_1, .., S_k], e = 0$`) =`
2   `pick` j `such that` $[\![e]\!]S_j \neq [\![e]\!]S_k$.
3   `pick` $q \in \mathbb{Q}$ `such that` $q \neq 0$ `and` $q \neq [\![e]\!]S_i$ `(for` $i = 1, \ldots k$`).`
4   `let` $\rho_0 = S_j \oplus_w S_k$`, where` $w = \frac{q - [\![e]\!]S_k}{[\![e]\!]S_j - [\![e]\!]S_k}$
5   `for` $i \leq k - 1$`,`
6   `let` S'_i `be the intersection of the plane`
    $e = 0$ `and the line passing through` ρ_0 `and` S_i
    `i.e.` $S'_i = S_i \oplus_{w_i} \rho_0$`, where` $w_i = \frac{q}{q - [\![e]\!]S_i}$.
7   `return` $[S'_1, .., S'_{k-1}]$.

There are a few details in the definition of the `Adjust` procedure that deserve discussion. Line 2 in the `Adjust` procedure presumes the existence of a point S_j in sample S such that the term e evaluates to distinct values at points S_j and S_k; this assumption is guaranteed by the pre-condition for the `Adjust` operation. (Geometrically, this means that the points S_j and S_k should lie at different distances from the plane $e = 0$.) The operation in line 3 is a linear time operation and the point ρ_0 is computed such that $[\![e]\!]\rho_0 = q$. Since we choose q, ρ_0 and S_i at different distances from the hyperplane $e = 0$, the line joining ρ_0 and S_i

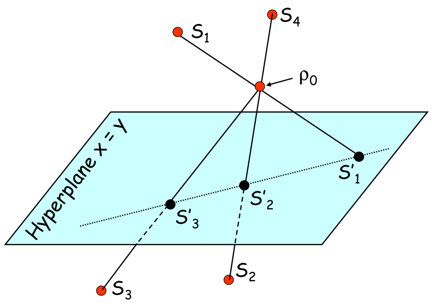


Fig. 2. An example of the `Adjust` procedure on a 4-point sample S, which satisfies the equality $z = x + y$. The adjustment is performed with respect to the equality $x = y$. The adjusted points S'_i are obtained as the intersections of the lines connecting the original points S_i with the point ρ_0. Note that the adjusted points lie on the line that represents the intersection of the hyperplanes $z = x + y$ and $x = y$.

intersects the hyperplane $e = 0$ in exactly one point. An example of the `Adjust` procedure is shown in Figure 2. The sample S consists of 4 points that lie in the plane $z = x + y$. We pick the point S_2 to play the role of S_j (where j is as in line 2) since the line passing through S_2 and S_4 is not parallel to the plane $x = y$. We then pick another point ρ_0 on the line passing through S_2 and S_4 such that it does not lie in the plane $x = y$ and the lines passing through it and any other other point in S are not parallel to the plane. Then, we obtain the points $S'_i(i = 1, 2, 3)$ as the intersection of the the lines that pass through ρ_0 and S_i with the plane $x = y$. Note that the resulting sample S' consists of 3 points that lie in the plane $x = y$ as well as the plane $z = x + y$.

We now prove that $S' = \texttt{Adjust}(S, e)$ has the desired properties A1, A2 and A3. We first state a useful and easily provable property of the affine combination operation.

Property 1 (Affine Combination Property). Let ρ_1 and ρ_2 be any two points, and let ρ_3 be any affine combination of ρ_1 and ρ_2. If ρ_1 and ρ_2 satisfy any linear equality $e = 0$, then ρ_3 also satisfies the equality $e = 0$.

It follows from Property 1 that if all points in sample S satisfy some equality $t = 0$, then so does ρ_0 (since it is an affine combination of some two points in sample S) and any point in sample S' (since it is an affine combination of ρ_0 and some point in sample S). Thus, sample S' has property A1. The points in sample S' lie on the hyperplane $e = 0$ (by definition), and hence $S' \models e = 0$. Thus, sample S' has property A2. For $i \leq k - 1$, we have $S'_i = S_i \oplus_{w_i} \rho_0$. Note that this means that there is a value w'_i such that $S_i = S'_i \oplus_{w'_i} \rho_0$. Also S_k can be expressed as an affine combination of S'_j and ρ_0. This means that S satisfies all the linear equalities satisfied by both S' and ρ_0. In order to show that S' has property A3, we assume that $S' \models t = 0$ and we show that $S \models t + \lambda e = 0$, for

$\lambda = -\frac{[\![t]\!]\rho_0}{[\![e]\!]\rho_0}$. Since $S' \models e = 0$, we have that $S' \models t + \lambda e = 0$. It is easy to verify that $\rho_0 \models t + \lambda e = 0$. Thus, $S \models t + \lambda e = 0$. Hence, sample S' has property A3.

3.2 The Satisfiability Procedure

The `IsSatisfiable` procedure described below is a randomized algorithm that takes as input a formula ϕ and a r-point random sample R. The only random choice in this algorithm is the value of this initial sample R. If ϕ is unsatisfiable, the algorithm returns `false` for any choice of R. If ϕ is satisfiable, the algorithm returns `true` with high probability over the choice of the random sample R.

1 `IsSatisfiable`(ϕ, R) =
2   `let` ϕ `be` $\{t_i = 0\}_{i=1}^{k} \cup \{t'_j \neq 0\}_{j=1}^{m}$
3   $S \leftarrow R$
4   `for` i `=` 1 `to` k`:`
5   `if` $S \models t_i + c = 0$ `for some` $c \neq 0$`, then return false`
6   `else if` $S \not\models t_i = 0$ `then` $S \leftarrow$ `Adjust`$(S, t_i = 0)$
7   `for` j `=` 1 `to` m`:`
8   `if` $S \models t'_j = 0$`, then return false`
9   `return true`

The loop starting in line 4 adjusts the sample incrementally so that it satisfies each equality in turn. Finally, the loop starting in line 7 checks for each disequality if there is an assignment in the resulting sample that satisfies it.

We now state the completeness and soundness results for this algorithm. Then, in Section 4 we show how to extend this algorithm to handle uninterpreted function symbols as well.

Theorem 1 (Completeness Theorem). *If* `IsSatisfiable`*(ϕ, R) returns true, then ϕ is satisfiable.*

Proof. Suppose `IsSatisfiable`(ϕ, R) returns true. Due to properties A1 and A2 of the `Adjust` operation, at the end of the loop starting in line 4 we have an adjusted sample S whose assignments satisfy all the equality literals of the formula. We know from linear algebra that the formula ϕ is satisfiable if and only if all of the formulas $\{t_1 = 0\}_{i=1}^{k} \cup \{t'_j \neq 0\}$ (for $j = 1, \ldots, m$) are satisfiable. The loop starting in line 7 ensures that each such formula is satisfied by at least one assignment in the final sample.

Theorem 2 (Soundness Theorem). *If ϕ is satisfiable, then* `IsSatisfiable`(ϕ, R) *returns true with high-probability over the random choice of the initial sample R.*

In order to prove the soundness theorem, we define the notion of *consistency of a sample with a formula*:

Definition 1. *Given a formula ϕ and a sample S, we say that S is* consistent *with ϕ if*

$$\phi \text{ is satisfiable} \Rightarrow (\forall t.S \models t = 0 \Rightarrow \phi \cup \{t = 0\} \text{ is satisfiable})$$

Intuitively, a sample S is consistent with a satisfiable formula ϕ, if S satisfies only those linear equalities that do not contradict ϕ. Note that any sample is consistent with an unsatisfiable formula. We have the following useful property.

Property 2. If S is consistent with the formula $\phi \cup \{e = 0\}$, then $\texttt{Adjust}(S, e = 0)$ is consistent with the same formula.

Proof. Assume that S is consistent with $\phi \cup \{e = 0\}$. Let $S' = \texttt{Adjust}(S, e = 0)$. Assume that $\phi \cup \{e = 0\}$ is satisfiable. Pick an arbitrary t such that $S' \models t = 0$. This means that $S \models t + \lambda e = 0$ (by property A3). Since S is consistent with $\phi \cup \{e = 0\}$, we know that $\phi \cup \{e = 0, t + \lambda e = 0\}$ must be satisfiable. Consequently $\phi \cup \{e = 0, t = 0\}$ must be satisfiable. This completes the proof.

Using Property 2, we can easily prove the following lemma:

Lemma 1. *If the initial random sample R is consistent with ϕ, and* `IsSatisfiable`*(ϕ, R) returns false, then ϕ is unsatisfiable.*

Proof. Suppose that the initial sample is consistent with ϕ. It follows from Property 2 that the sample S in procedure `IsSatisfiable` always remains consistent with ϕ. Now, consider the following two cases.

- Suppose `IsSatisfiable` returns `false` in line 5. Then $S \models t_i + c = 0$. Since S is consistent with ϕ and $t_i + c = 0 \cup \{\phi\}$ is unsatisfiable, it must be that ϕ is unsatisfiable.
- Suppose `IsSatisfiable` returns `false` in line 8. Then $S \models t'_i = 0$. Since S is consistent with ϕ, and also $\phi \cup \{t'_i = 0\}$ is unsatisfiable, it must be that ϕ is unsatisfiable.

This means that as long as we start with a sample that is consistent with the input formula ϕ, the algorithm is sound. The question now is how to choose the initial sample such that it is consistent with any given formula ϕ. The key observation is that we can choose R randomly because there are many more samples that are consistent with ϕ than those that are not. This is obvious if ϕ is unsatisfiable, because then all samples are consistent with ϕ. If ϕ is satisfiable, R is inconsistent with ϕ only if there is a term t such that $\phi \Rightarrow t \neq 0$ and $R \models t = 0$. Such a term t can be written as a linear combination of the equality literals of ϕ added to either the constant 1 or one of the disequality literals of ϕ. For any such term t, it is unlikely that we choose R such that all of its r assignments satisfy $t = 0$. The following lemma provides an upper bound on the probability that a randomly chosen sample R is inconsistent with a formula ϕ.

Lemma 2 (Consistent Random Sample Lemma). *If ϕ is satisfiable, then the probability that the r-point random sample R is inconsistent with ϕ is at* *most* $(m+1)\frac{|F|}{|F|-3r}\left(\frac{3r}{|F|}\right)^{r-k'}$*, where* m *is the number of disequality literals in* ϕ*,* $|F|$ *is the size of the finite subset of* $\mathbb{Q}$ *from which we choose the elements of* R *uniformly at random and independently of each other, and* $k' \le k$ *is the maximum number of linearly independent equality literals in* ϕ*.*

This lemma along with Lemma 1 proves Theorem 2 and also provides an upper bound for the probability that our satisfiability algorithm incorrectly reports a satisfiable formula to be unsatisfiable.

The proof of Lemma 2 is somewhat involved and is given in the Appendix. Note that the probability of error increases linearly with the number of disequalities (because we might make an independent error in handling each one of them). The dominant factor is $\left(\frac{3r}{|F|}\right)^{r-k'}$, which decreases with the size of the subset from which we make random choices. (We cannot choose directly from $\mathbb{Q}$ because each choice would need an infinite number of random bits.) The probability of error also decreases exponentially when we increase r. Essentially, when we work with more random assignments it becomes less likely that all of them accidentally satisfy an equality. The `IsSatisfiable` algorithm performs at most k `Adjust` operations, one for each equality literal in ϕ. However, the `Adjust` operation is performed only if the equality literal is not entailed by the previously processed equalities. This means that `Adjust` is performed only k' times. The $r-k'$ exponent suggests that r should be at least as large as k'. This makes sense because we have seen that each `Adjust` operation "looses" one assignment.

4 Extension to Uninterpreted Function Symbols

We now extend the satisfiability procedure to handle formulas that also contain uninterpreted function symbols. We first introduce some notation.

For any term t, let $V(t)$ be the term obtained from t by replacing all occurrences of the outermost function term by a fresh variable as follows: $V(t_1+t_2)=V(t_1)+V(t_2)$, $V(t_1-t_2)=V(t_1)-V(t_2)$, $V(f(t_1,\ldots,t_k))=v_{f(t_1,..,t_k)}$, $V(q\times t)=q\times V(t)$, $V(q)=q$. Let $\mathcal{C}(\phi)$ denote the formula obtained from ϕ after performing the Ackerman transformation [1] as follows: (1) each term t in ϕ is replaced by $V(t)$, and (2) for every pair of distinct function terms $f(t_{1,1},..,t_{1,k})$ and $f(t_{2,1},..,t_{2,k})$ in ϕ, we introduce the *conditional equality* $(\bigwedge_{i=1,..,k} V(t_{1,i})=V(t_{2,i}))\Rightarrow(V(f(t_{1,1},..,t_{1,k}))=V(f(t_{2,1},..,t_{2,k})))$. Following is an example of a formula ϕ and the corresponding $\mathcal{C}(\phi)$:

$$\begin{aligned}\phi\quad &= \{f(x+3)=f(z), f(y+x)=y, y=3\}\\ \mathcal{C}(\phi) &= \{v_1=v_2, v_3=y, y=3, (x+3=z)\Rightarrow(v_1=v_2),\\ &\qquad (x+3=y+x)\Rightarrow(v_1=v_3), (z=y+x)\Rightarrow(v_2=v_3)\}\end{aligned}$$

Here we have introduced new variables v_1, v_2 and v_3 for the terms $f(x+3), f(x+y)$ and $f(y+x)$ respectively. The conditional equalities that are used to obtain

$\mathcal{C}(\phi)$ from ϕ capture the essence of the congruence axiom for uninterpreted functions, and one can easily show that ϕ is satisfiable if and only if $\mathcal{C}(\phi)$ is satisfiable.

For any formula ϕ, let $\mathcal{A}(\phi)$ be the formula that does not contain any uninterpreted function symbols or conditional equalities, and is obtained from $\mathcal{C}(\phi)$ as follows. Each conditional equality of the form $(\bigwedge_{i=1,..,k} s_i = s'_i) \Rightarrow (v = v')$ in $\mathcal{C}(\phi)$ is replaced with the equality $v = v'$ if $\mathcal{C}(\phi) \Rightarrow s_i = s'_i$ for all $i = 1, .., k$, or with the disequality $v \neq v'$ otherwise. For the above example, we have:

$$\mathcal{A}(\phi) = \{v_1 = v_2, v_3 = y, y = 3, v_1 = v_3, v_1 \neq v_2, v_2 \neq v_3\}$$

Just like $\mathcal{C}(\phi)$, $\mathcal{A}(\phi)$ is satisfiable if and only if ϕ is satisfiable. Note that $\mathcal{C}(\phi)$ is easy to compute but $\mathcal{A}(\phi)$ is not. This is not a problem because we use $\mathcal{A}(\phi)$ only in the correctness arguments.

The `IsSatisfiable'` procedure shown below decides the satisfiability of a formula ϕ by considering the modified formula $\mathcal{C}(\phi)$. The procedure makes use of a macro `Assume` that takes a sample and an equality literal as arguments, and has the following definition.

`Assume`$(S, t = 0)$ =
  `if` $S \models t + c = 0$ `for some` $c \neq 0$`, then return false`
  `else if` $S \not\models t = 0$`, then` $S \leftarrow$ `Adjust`$(S, t = 0)$

`IsSatisfiable'`(ϕ, R) =
  `let` $\mathcal{C}(\phi)$ `be` $\{t_i = 0\}_{i=1}^k \cup \{t'_i \neq 0\}_{i=1}^m \cup \{(\bigwedge_{j=1,..,k_i} s_{i,j} = s'_{i,j}) \Rightarrow v_i = v'_i\}_i^\ell$
  $S \leftarrow R$
  `for` i `= 1 to` k`:`
    `Assume`$(S, t_i = 0)$
    `repeat until no changes to` S `occur:`
      `for` w `= 1 to` ℓ`:`
        `if` $(\bigwedge_{j=1,..,k_w} S \models s_{w,j} - s'_{w,j} = 0)$`,` `Assume`$(S, v_w - v'_w = 0)$
    `for` i `= 1 to` m`:`
      `if` $S \models t'_i = 0$`, then return false`
    `return true`

Note that `IsSatisfiable'`(ϕ, R) returns the correct answer if and only if `IsSatisfiable`$(\mathcal{A}(\phi), R)$ returns the correct answer. It follows from Theorem 1 that if ϕ is unsatisfiable, then `IsSatisfiable'`(ϕ, R) returns `false`. It also follows from Theorem 2 that if ϕ is satisfiable, then `IsSatisfiable'`(ϕ, R) returns `true` with probability (over the random choices for the r-point sample R) at least $1 - (m' + 1)\frac{|F|}{|F|-3r}(\frac{3r}{|F|})^{r-k'}$, where m' is the number of disequality literals in $\mathcal{A}(\phi)$, and k' is the maximum number of linearly independent equality literals in $\mathcal{A}(\phi)$. Clearly, $m' \leq m + \ell^2$, where m is the number of disequality literals in ϕ and ℓ is the number of function terms in ϕ. Also, $k' \leq k + \ell$ since there can be at most ℓ linearly independent equalities among ℓ function terms.

The `IsSatisfiable'` algorithm as presented here emphasizes logical clarity over efficiency. In our experiments, we use an optimized variant of this algorithm that does not create the conditional equalities in $\mathcal{C}(\phi)$ explicitly. Instead, we maintain, for each function symbol f, a list of pairs of the form $([s_1,..,s_k],v)$ for each function term $f(t_1,..,t_k)$, where $s_i = V(t_i)$ and $v = V(f(t_1,..,t_k))$. For our example, the list corresponding to f is $\{([x+3],v_1),([z],v_2),([y+x],v_3)\}$. This allows us to find quickly, in line 7, the pairs of $[s_1,..,s_k]$ and $[s'_1,..,s'_k]$ such that $S \models s_j - s'_j = 0$ for all $j = 1,..,k$, by using a hash table indexed by $[\,[\![s_1]\!]S_1,..,[\![s_k]\!]S_1\,]$, i.e. the values of the terms s_j at the point S_1.

5 Retracting Assumptions

It is often the case that we must solve a number of satisfiability problems that share literals. Such a situation arises naturally in the context of program verification when the formulas correspond to paths and are constructed as conjunction of branch conditions. For example, consider the program fragment:

```
if z = x + y then
   if x = y then assert (z = 2x) else assert (x = z - y)
```

This fragment can be verified by checking the unsatisfiability of the two formulas $\{z = x+y, x = y, z \neq 2x\}$ and $\{z = x+y, x \neq y, x \neq z-y\}$. If we process these formulas independently, we end up duplicating work for *assuming* $z = x+y$. Instead, if we have a satisfiability procedure that can retract assumptions, then after processing the first formula we can retract the equality $x = y$ and continue with the disequalities in the second formula.

Another situation where ability to retract assumptions is important is the context of a Nelson-Oppen theorem prover [6], in which non-convex theories are handled using backtracking. Similarly, a Shostak [8] theorem prover handles non-solvable theories using backtracking.

In our algorithm, a naive way to retract the last equality assumption is to restore the current sample to the sample before the `Adjust` operation. One method to do this is to remember the old samples, but this has a high space overhead. Another method relies on the fact that we can recover the previous sample S from the adjusted one, if we remember just the weights w_i.

Next we show a technique better than the above mentioned methods. The key observation is that we need not restore the original sample exactly, as long as we obtain an equivalent sample in the sense that it satisfies exactly the same linear equalities as the original one. To achieve this we extend the `Adjust` operation to return not just an adjusted sample but also a point that when added to the adjusted sample produces a sample equivalent to the original one. This means that we need to remember only this special point and we can undo an `Adjust` operation by adding this special point to the adjusted sample.

5.1 The `Adjust′` Operation

Let `Adjust′` be the operation that takes a sample S and a term e as input, where $S \not\models e + c = 0$ for any constant $c \neq 0$, and returns another sample S' and a point ρ. The adjusted sample S' satisfies the properties A1, A2, A3 mentioned in Section 3.1, and the point ρ satisfies the following additional properties:

B1. For any term t, if $S \models t = 0$ then $\rho \models t = 0$.
B2. For any term t, if $S' \models t = 0$ and $\rho \models t = 0$ then $S \models t = 0$.

These properties, along with property A1, mean that S satisfies exactly the same linear equalities that are satisfied by both S' and ρ.

An Implementation of the `Adjust′` Operation We now present an efficient implementation of the `Adjust′` operation:

```
1 Adjust′(S, e) =
2   let S′ ← Adjust(S,e).
3   pick j such that S_j ⊭ e = 0.
4   return (S′, S_j)
```

The precondition for `Adjust′` ensures that an appropriate j can be found in line 3. It is a simple exercise to verify that $(S', \rho) =$ `Adjust′`(S, e) satisfies the properties A1, A2, A3, B1, and B2.

5.2 The `UnAdjust` Operation

The modified satisfiability procedure is just like the one described in Section 4 except that it uses the `Adjust′` operation in place of the `Adjust` operation and remembers the point ρ returned by the `Adjust′`.

We now define the operation `UnAdjust` for retracting the last equality that was adjusted for. The operation `UnAdjust` takes the current sample S and the point ρ corresponding to the last equality and returns another sample S' such that S' satisfies exactly those linear equalities that are satisfied by both S and ρ. The `UnAdjust` operation can be implemented efficiently as

$$\texttt{UnAdjust}([S_1, \ldots, S_k], \rho) = [S_1, \ldots, S_k, \rho].$$

5.3 Correctness of Retraction

Consider the algorithm `IsSatisfiable′`. We must retract assumptions in the reverse order in which they were made. In order to retract an assumption $t_i = 0$, we must invoke `UnAdjust` for all of the `Adjust` operations that are performed in the i^{th} iteration of the loop starting at line 4.

Lemma 3 (The Adjust-UnAdjust Lemma). *Let* $(S_1, \rho) =$ `Adjust′`$(S_0, e = 0)$ *and* S_2 *a sample that satisfies the same linear equalities as* S_1*, and* $S_3 =$ `UnAdjust`(S_2, ρ)*. Then* S_3 *satisfies the same linear equalities as* S_0*.*

Proof. Let t be an arbitrary term. We first prove that if $S_0 \models t = 0$ then $S_3 \models t = 0$. Due to property A1 we know that $S_1 \models t = 0$ and thus $S_2 \models t = 0$. Due to property B1, we know that $\rho \models t = 0$ and hence from the definition of `UnAdjust` we conclude that $S_3 \models t = 0$. Next we prove that if $S_3 \models t = 0$ then $S_0 \models t = 0$. From definition of `UnAdjust` we know that $S_2 \models t = 0$ and $\rho \models t = 0$. Hence $S_1 \models t = 0$, and from property B2 $S_0 \models t = 0$.

It follows from Lemma 3 that if a sample S is consistent with a formula ϕ, then the sample obtained from S after any number of `Adjust` and an equal number of corresponding `UnAdjust` operations is also consistent with ϕ.

6 Experimental Results

We have implemented the `IsSatisfiable`$'$ procedure described in Section 4 in C with some optimizations. One important optimization that we have used is to perform arithmetic operations over the field $\mathcal{Z}_p$ for some randomly chosen prime p. This avoids the need to perform arbitrary precision arithmetic, which is otherwise required if the operations are over rational numbers. This optimization is problematic in an otherwise-deterministic algorithm, but for our randomized algorithm it simply results in an additional probability of error. For lack of space, we do not present the analysis of the error probability that results from working over $\mathcal{Z}_p$ rather than $\mathbb{Q}$. This idea is similar to fingerprinting mechanisms that involve performing arithmetic modulo a randomly chosen prime [5].

We compared the running-time performance of our implementation with the SRI's ICS (Integrated Canonizer and Solver) decision procedure package [3], which is implemented in Ocaml. ICS is directly based on the refinement of Shostak's algorithm by Ruess and Shankar [7]. The implementation of ICS uses optimization techniques such as hash-consing and efficient data structures like Patricia trees for representing sets and maps efficiently. ICS uses arbitrary precision rational numbers from the GNU multi-precision library (GMP).

Figure 3 shows the time (excluding the parsing time) in milliseconds taken by our implementation and ICS on several examples. Column *Rand* shows the time taken by our implementation when run with the best possible parameters, which include working with as few points as required, and performing arithmetic operations over a small field (in this case $Z_{268435399}$, so that the arithmetic can be performed using 32-bit integers). The experiments were performed on a 1.7 GHz Pentium 4 machine running Linux 2.4.5. The examples used in our experi-

Example	#equalities	ICS (ms)	Rand (ms)	ICS/Rand	#points	#adjusts
arith-dense	26	386.4	2.3	168.0	30	25
arith-sparse	25	84.8	1.3	65.2	20	14
both-dense	20	37.0	3.4	10.8	40	29
both-sparse1	50	73.9	7.5	9.9	50	42
both-sparse2	150	165.0	20.2	8.2	60	55
both-sparse3	300	325.3	51.0	6.4	70	80
uf-single	5	1.7	0.7	2.4	20	16
uf-copies	25	4.0	1.0	4.0	20	16
uf-wrong	35	23.1	9.6	2.4	40	72

Fig. 3. This table compares the time (in milliseconds) taken by our implementation and ICS on several examples. Column *ICS* shows the time taken by ICS, while column *Rand* shows the time taken by our implementation when run with the number of points mentioned in column *#points*. Column *#adjusts* denotes the number of `Adjust` operations performed by our implementation. Column *#equalities* denotes the number of equality literals.

ments can be classified based on the size, number, and type of equality literals, and the number of equivalence classes on terms and sub-terms of the formula. The first two examples, *arith-dense* and *arith-sparse*, involve only linear arithmetic. *arith-dense* contains equalities, each of which has many sub-terms, while *arith-sparse* contains equalities with a small number of sub-terms in each literal. The next four examples, *both-dense*, *both-sparse1*, *both-sparse2* and *both-sparse3* involve both linear arithmetic and uninterpreted functions. Of these *both-dense* contains dense equalities, while the rest contain sparse equalities. All of these examples were generated randomly. The next three examples *uf-single*, *uf-copies* and *uf-wrong* involve only uninterpreted functions, and have been taken from the paper by Bachmir and Tiwari [2] that compared the performance of several congruence closure algorithms. *uf-single* contains five equalities that induce a single equivalence class. *uf-copies* is same as *uf-single*, except that it contains five copies of all the equalities. *uf-wrong* consists of equalities that result in many `Adjust` operations. Note that for some examples, #adjusts is less than #equalities. This is because of the following optimization that we use. While processing an equality literal, if a variable is encountered for the first time, then instead of adjusting the current sample, we simply set the value of that variable in the sample such that the equality literal is satisfied. Also note that for some examples, #points is less than #adjusts. This is because of another optimization where for each equality, we *adjust* only those variables that are *dependent* on some variable in that equality. This allows us to work with fewer points than the number of adjust operations.

Following observations can be drawn from our experiments. Our algorithm has a maximum speed-up over the ICS package when there is more arithmetic involved. This is expected since randomization helps to reason about the arithmetic faster. When the equalities get sparser, and contain more uninterpreted functions, the speed-up of our algorithm over ICS decreases.

Working with more points reduces the error probability, but increases the running-time. Theoretically, the running-time of our algorithm grows quadratically with the number of points in the initial random sample. Fortunately, the error probability decreases exponentially with the number of points.

7 Related Work

A notable difference between the algorithm that we have described here and the existing deterministic algorithms that solve a similar problem is the handling of the arithmetic. Instead of manipulating symbolic expressions we simply evaluate the arithmetic expression. This is a simpler operation and even gives us a slight advantage in the presence of non-linear arithmetic. For example, our algorithm can very naturally prove the unsatisfiability of the formula $x = y \wedge x^2 - 2xy + y^2 \neq 0$. However, the advantage is slight because the `Adjust` operation we have does not work with non-linear equalities, which means that we can handle non-linearity only in the disequalities and as arguments to uninterpreted function symbols.

The existing deterministic algorithms for the combination of linear equalities and uninterpreted function symbols are typically constructed from two separate satisfiability procedures for the two theories, along with a mechanism for combining satisfiability procedures. One such mechanism is described by Nelson and Oppen [6] and requires the individual satisfiability procedures to communicate only equalities between variables. Our algorithm has a similar communication mechanism, specifically implemented by the loop in line 6 in the definition of `IsSatisfiable`′. The difference is that we detect an equality between terms when they have equal values in all the random assignments.

Shostak [8] gave a more efficient algorithm, which works for the theory of uninterpreted functions and for solvable and canonizable theories. The theory of linear arithmetic is canonizable and solvable. A canonizer σ for linear arithmetic must rewrite terms into an ordered sum-of-monomials form. A solver for linear arithmetic may take an equality of the form $c + \sum_{i=1}^{n} a_i x_i = 0$ and return $x_1 = \sigma(-\frac{c}{a_1} + \sum_{i=2}^{n} -\frac{a_i}{a_1})$, where $a_1 \neq 0$. The ICS tool that we have used in our performance comparisons uses Shostak's algorithm.

There are similarities between Shostak's algorithm and our randomized algorithm. Our `Adjust` operation is similar to the solve procedure used in Shostak's algorithm since both serve the purpose of propagating a new equality. The sample S maintained by the randomized algorithm at each step can be regarded as a canonizer, since for any term t, $[[\![t]\!]S_1, \ldots, [\![t]\!]S_r]$ is a *probabilistic* canonical form for t in the following sense. Two terms that are congruent have the same canonical form, while there is a small probability that two non-congruent terms have the same canonical forms.

The soundness of Shostak's algorithm is straightforward, but its completeness and termination have resisted proofs for a couple of decades. Shostak's original algorithm and several of its subsequent variations are all incomplete and potentially non-terminating. Recently, Ruess and Shankar [7] have presented a correct version of the algorithm along with rigorous proofs for its correctness. Similar difficulties in carrying out the correctness proof seem to arise for randomized algorithms, but here the difficulties are not due to the complexity of the algorithm but due to the complexity of probability analysis. This is typical of randomized algorithms, which are usually easy to describe, simple to implement, but require subtle proofs to bound the error probability.

There are similarities between this randomized algorithm and the random interpretation that we have described in an earlier paper [4] for the purpose of discovering linear equalities in a program. The contributions of this paper are a modified `Adjust` algorithm that also handles uninterpreted function symbols and allows for retracting assumptions, and a more general proof of soundness. In our earlier paper the proof of probabilistic soundness relies on the fact that the analysis is performed over a finite field. In this paper, mostly because the application domain is simpler, we are able to give a different proof that does not rely on the finiteness of the field over which the satisfiability is checked.

8 Conclusion and Future Work

We have described a randomized algorithm for deciding the satisfiability of a conjunction of equalities and disequalities involving linear arithmetic and uninterpreted function symbols. The most notable feature of this algorithm is simplicity of its data structures and of the operations it performs. The cost for this simplicity is that, in rare occasions, it might incorrectly decide that a satisfiable formula is not satisfiable. However, we have shown that the probability that this happens is very small and can be controlled by varying the number of points in the initial random sample or the size of the set from which the random values are chosen. Thus, the error probability can be reduced to an infinitesimally small value so that it is negligible for all practical purposes.

An interesting direction for future work is to explore whether these ideas can be extended to other theories, such as inequalities, or arrays. One possible approach is suggested by the observation that when we evaluate terms in a random sample we essentially compute a hash value for the term, such that if two terms have the same hash values then, with high probability, they are equal. For arithmetic this is naturally achieved by just performing arithmetic on some random inputs. Perhaps we can find similar hash functions for other theories. Another promising direction for future research is integration of symbolic techniques with randomized ones.

References

1. W. Ackermann. *Solvable Cases of the Decision Problem.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1954.
2. L. Bachmair and A. Tiwari. Abstract congruence closure and specializations. In D. McAllester, editor, *Conference on Automated Deduction, CADE '2000*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 64–78, Pittsburgh, PA, June 2000. Springer-Verlag.
3. J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonization and Solving. In G. Berry, H. Comon, and A. Finkel, editors, *Computer-Aided Verification, CAV '2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249, Paris, France, July 2001. Springer-Verlag.
4. S. Gulwani and G. C. Necula. Discovering affine equalities using random interpretation. In *The 30th Annual ACM Symposium on Principles of Programming Languages*, pages 74–84. ACM, Jan. 2003.
5. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
6. G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, Oct. 1979.
7. H. Ruess and N. Shankar. Deconstructing Shostak. In *16th Annual IEEE Symposium on Logic in Computer Science (LICS '01)*, pages 19–28, Washington - Brussels - Tokyo, June 2001. IEEE.
8. R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, Jan. 1984.

A Proof of Consistent Random Sample Lemma

Lemma 2 (Consistent Random Sample Lemma). *If ϕ is satisfiable, then the probability that the r-point random sample R is inconsistent with ϕ is at most $(m+1)\frac{|F|}{|F|-3r}\left(\frac{3r}{|F|}\right)^{r-k'}$, where m is the number of disequality literals in ϕ, $|F|$ is the size of the finite subset of $\mathbb{Q}$ from which we choose the elements of R uniformly at random and independently of each other, and k' is the maximum number of linearly independent equality literals in ϕ.*

Proof. Without any loss of generality, let us assume that $\{t_i = 0\}_{i=1}^{k'}$ is any maximal set of linearly independent equalities in the satisfiable formula ϕ. Then R is *not* consistent with ϕ iff there exists a t such that $\phi \Rightarrow t \neq 0$ and $R \models t = 0$. It follows from linear algebra that t can be written as a linear combination of t_i (for $i = 1,..,k'$) added to either the constant 1 or one of t'_j (where $j \in \{1,..,m\}$). The error probability for each of these $m+1$ cases can be obtained by instantiating t in the following Lemma 4 with either the constant 1 or one of the t'_j. The desired bound on the probability of error can now be obtained by multiplying the probability of error in each case by $m+1$.

Lemma 4. *Let $t_1,..,t_{k'}$ be linearly independent terms in variables $x_1,..,x_n$ and t an additional term, such that the formula $\{t_j = 0\}_{j=1}^{k'} \cup \{t \neq 0\}$ is satisfiable. Then,*

$$\Pr_R[\exists \alpha_1,..,\alpha_{k'} \text{ such that } R \models (t + \sum_{i=1}^{k'} \alpha_i t_i = 0)] \leq \frac{|F|}{|F|-3r}\left(\frac{3r}{|F|}\right)^{r-k'}.$$

Proof. Let $\mathcal{E}$ be the event that there exist $\alpha_1,..,\alpha_{k'}$ such that $R \models (t + \sum_{i=1}^{k'} \alpha_i t_i = 0)$. Let $\mathcal{L}$ be the following system of r equations in variables $z_1,..,z_{k'}$:

$$\{[\![t]\!]R_j + \sum_{i=1}^{k'} ([\![t_i]\!]R_j) z_i = 0\}_{j=1}^{r}.$$

Event $\mathcal{E}$ occurs if and only if $\mathcal{L}$ has a solution. Let $C_{r\times k'}$ and $\tilde{C}_{r\times(k'+1)}$ be the coefficient matrix and the augmented matrix[1] respectively for $\mathcal{L}$. $\mathcal{L}$ has a solution iff for all $i \in \{1,..,r\}$ if the i^{th} row of C is linearly dependent on the first $i-1$ rows of C, then the i^{th} row of $\tilde{C}$ is also linearly dependent on the first $i-1$ rows of $\tilde{C}$.

We partition the event $\mathcal{E}$ into cases depending on which set of rows in $\mathcal{C}$ are linearly independent of the previous rows. For any subset I of $\{1,..,r\}$, let $\mathcal{E}_I$ be the event that for any $i \in I$, the i^{th} row of C is linearly *independent* of the first $i-1$ rows of C, and for any $i \in \{1,..,r\} - I$, the i^{th} row of C is linearly *dependent* on the first $i-1$ rows of C. The set of events $\{\mathcal{E}_I \mid I \subseteq \{1,..,r\}, 1 \in I, |I| \leq k'\}$

[1] The augmented matrix is obtained from the coefficient matrix by adding a column corresponding to the constants.

is a disjoint partition of the underlying probability space since there can be at most k' linearly independent rows in $C_{r\times k'}$. Thus,

$$\Pr_R[\mathcal{E}] = \sum_{I\subseteq\{1,..,r\},1\in I,|I|\leq k'} \Pr_R[\mathcal{E}\cap\mathcal{E}_I] \tag{1}$$

It now follows from the Claim stated and proved below that

$$\Pr_R[\mathcal{E}\cap\mathcal{E}_{\mathcal{I}}] \leq \left(\frac{1}{|F|}\right)^{r-|I|} \tag{2}$$

Here is some intuition behind Equation 2. Note that the event $\mathcal{E}\cap\mathcal{E}_I$ occurs only when all the rows $d\in\{1,\ldots,r\}-I$ are linearly dependent on some rows in the set I, both in the coefficient matrix C and in the augmented matrix $\tilde{C}$. For each such row d, the probability of choosing the assignment R_d with elements from the finite set F such that this row is linearly dependent on the rows in I is at most $\frac{1}{|F|}$.

The desired probability for event $\mathcal{E}$ can now be obtained from Equations 1 and 2 as follows:

$$\begin{aligned}
\Pr_R[\mathcal{E}] &\leq \sum_{I\subseteq\{1,..,r\},1\in I,|I|\leq k'} \left(\frac{1}{|F|}\right)^{r-|I|}\\
&= \sum_{i\in\{1,..,k'\}} \binom{r-1}{i-1}\times\left(\frac{1}{|F|}\right)^{r-i}\\
&\leq \sum_{i\in\{1,..,k'\}} \left(\frac{(r-1)e}{r-i}\right)^{r-i}\times\left(\frac{1}{|F|}\right)^{r-i}\\
&\leq \sum_{i\in\{1,..,k'\}} \left(\frac{3r}{|F|}\right)^{r-i}\\
&\leq \frac{|F|}{|F|-3r}\times\left(\frac{3r}{|F|}\right)^{r-k'}
\end{aligned}$$

Claim. For any subset I of $\{1,..,r\}$, $\Pr_R[\mathcal{E}\cap\mathcal{E}_{\mathcal{I}}] \leq \left(\frac{1}{|F|}\right)^{r-|I|}$.

Proof. For any subset I of $\{1,..,r\}$ and for any $i\in I$, let $\mathcal{F}_{I,i}$ be the event that the i^{th} row of C is linearly independent of the first $i-1$ rows of C. For any subset I of $\{1,..,r\}$ and for any $i\in\{1,..,r\}-I$, let $\mathcal{G}_{I,i}$ be the event that the i^{th} row of C is linearly dependent on the first $i-1$ rows of C, and let $\tilde{\mathcal{G}}_{I,i}$ be the event that the i^{th} row of $\tilde{C}$ is linearly dependent on the first $i-1$ rows of $\tilde{C}$. By definition of event $\mathcal{E}_I$, for any subset I, the event $\mathcal{E}_I$ occurs iff the events $\{\mathcal{F}_{I,i}\}_{i\in I}$ and the events $\{\mathcal{G}_{I,i}\}_{i\in\{1,..,r\}-I}$ occur. Thus, $\Pr_R[\mathcal{E}_I] = \Pr_R[\bigwedge\limits_{i\in I}\mathcal{F}_{I,i} \ \wedge \bigwedge\limits_{i\in\{1,..,r\}-I}\mathcal{G}_{I,i}]$.

Let I be any subset of $\{1,..,r\}$ such that $1\in I$ and I contains at most k' elements. It follows from the definition of $\tilde{\mathcal{G}}_{I,i}$ and the necessary and sufficient condition for event $\mathcal{E}$ mentioned at the end of the first paragraph in the proof of Lemma 4 that

$$\begin{aligned}
\Pr_R[\mathcal{E} \cap \mathcal{E}_I] &= \Pr_R[\bigwedge_{i \in I} \mathcal{F}_{I,i} \quad \wedge \bigwedge_{i \in \{1,..,r\}-I} \tilde{\mathcal{G}}_{I,i}] \\
&= \prod_{i \in I} \Pr_R[\mathcal{F}_{I,i} \mid \bigwedge_{j \in I, j<i} \mathcal{F}_{I,j} \quad \wedge \bigwedge_{j \in \{1,..,r\}-I, j<i} \tilde{\mathcal{G}}_{I,j}] \\
&\quad \times \prod_{i \in \{1,..,r\}-I} \Pr_R[\tilde{\mathcal{G}}_{I,i} \mid \bigwedge_{j \in I, j<i} \mathcal{F}_{I,j} \quad \wedge \bigwedge_{j \in \{1,..,r\}-I, j<i} \tilde{\mathcal{G}}_{I,j}] \\
&\leq \prod_{i \in \{1,..,r\}-I} \Pr_R[\tilde{\mathcal{G}}_{I,i} \mid \bigwedge_{j \in I, j<i} \mathcal{F}_{I,j} \quad \wedge \bigwedge_{j \in \{1,..,r\}-I, j<i} \tilde{\mathcal{G}}_{I,j}] \qquad (3)
\end{aligned}$$

For any $i \in \{1, .., r\} - I$, let I_i be the set $\{j \in I \mid j < i\}$ and let $n_i = |I_i|$. Let $M_{n_i \times k'}$ be the sub-matrix of C that consists of the rows of C with indices from set I_i. Let $\tilde{M}_{(n_i+1)\times(k'+1)}$ be the sub-matrix of A that consists of the rows of A with indices from set $I_i \cup \{i\}$.

Consider any $i \in \{1, .., r\} - I$. We now bound the quantity $\Pr_R[\tilde{\mathcal{G}}_{I,i} \mid \bigwedge_{j \in I, j<i} \mathcal{F}_{I,j} \quad \wedge \bigwedge_{j \in \{1,..,r\}-I, j<i} \tilde{\mathcal{G}}_{I,j}]$. Suppose that the assignments $R_1, .., R_{i-1}$ have been chosen such that the events $\{\mathcal{F}_{I,j}\}_{j \in I_i}$ and the events $\{\tilde{\mathcal{G}}_{I,j}\}_{j \in \{1,..,i-1\}-I_i}$ occur, and we have to choose the assignment R_i. Since the events $\{\mathcal{F}_{I,j}\}_{j \in I_i}$ occur, the rows in M are linearly independent, i.e. $Rank(M) = n_i$. Thus, there exists a sub-matrix $T_{n_i \times n_i}$ of M such that $Rank(T) = n_i$, i.e. $Det(T) \neq 0$. Let $T'_{(n_i+1)\times(n_i+1)}$ be the sub-matrix of M' that has T as a sub-matrix and an additional row corresponding to the i^{th} row of A and an additional column that contains all $1's$. Since the events $\{\tilde{\mathcal{G}}_{I,j}\}_{j \in \{1,..,r\}-I_i}$ occur, the event $\tilde{\mathcal{G}}_{I,i}$ occurs iff the assignment R_i is chosen such that the i^{th} row of $\mathcal{A}$ turns out to be linearly dependent on the rows of A with indices from set I_i, which implies that $Rank(\tilde{T}) = n_i$, or, equivalently, $Det(\tilde{T}) = 0$. Since we have not yet chosen the assignment R_i, $Det(\tilde{T})$ is a linear multivariate polynomial in variables $x_1, .., x_n$. Note that $Det(\tilde{T})$ is not identically equal to 0 because otherwise we can write 1 as a linear combination of the terms $t_1, .., t_{k'}$ (expand the determinant with respect to the row not present in sub-matrix T) which will contradict the assumption that $\{t_j = 0\}_{j=1}^{k'}$ is satisfiable. The probability that some polynomial of degree 1 that is identically not equal to zero, evaluates to zero when the values for its variables are chosen independently and u.a.r. from the set F is at most $\frac{1}{|F|}$. Thus,

$$\Pr_R[\tilde{\mathcal{G}}_{I,i} \mid \bigwedge_{j \in I, j<i} \mathcal{F}_{I,j} \quad \wedge \bigwedge_{j \in \{1,..,r\}-I, j<i} \tilde{\mathcal{G}}_{I,j}] \leq \frac{1}{|F|} \qquad (4)$$

The required result now follows from Equations 3 and 4.