A Randomized Satisfiability Procedure for Arithmetic and Uninterpreted Function Symbols [1]

Sumit Gulwani , George C. Necula

Department of Computer Science, UC-Berkeley, Berkeley, CA 94720, USA

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 algorithm can also be easily adapted to produce proofs for its output. 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.

Key words: Randomized Algorithm, Satisfiability Procedure, Linear Arithmetic, Uninterpreted Function Symbols

Email addresses: `gulwani@cs.berkeley.edu` (Sumit Gulwani), `necula@cs.berkeley.edu` (George C. Necula).
URLs: `http://www.cs.berkeley.edu/~gulwani` (Sumit Gulwani), `http://www.cs.berkeley.edu/~necula` (George C. Necula).

[1] 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.

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 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.

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

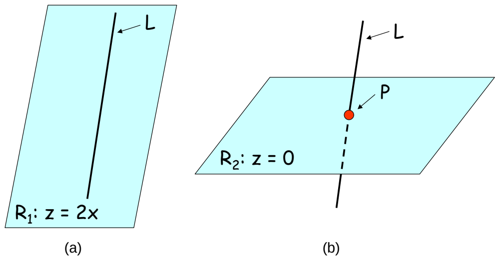

Fig. 1. The line L represents the solution space for the constraint $(z = x + y) \wedge (x = y)$. If we choose points randomly on L, we can easily deduce that $L \Rightarrow R_1$ and $L \not\Rightarrow R_2$.

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 instead of 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). We then extend the algorithm to handle uninterpreted function symbols in Section 4. 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 a procedure to produce a proof for the output of the algorithm by using some information computed by the algorithm; such a certificate can be used to remove any error probability from the algorithm. In Section 7, we describe an interesting optimization, namely a randomized transformation for converting arbitrary function terms to terms with only one unary function symbol; such an optimization makes the implementation of the algorithm simpler and efficient. In Section 8, 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 t 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 assignment ρ (using the usual interpretation of 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} assignment in 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. Similarly, we define an affine combination of m assignments $\rho_1, \ldots, \rho_m$ with weights $w_1, \ldots, w_{m-1} \in \mathbb{Q}$ (denoted by $\rho_1 \oplus_{w_1} \ldots \oplus_{w_{m-1}} \rho_m$) as another assignment ρ such that for any variable x, $\rho(x) = w_1 \times \rho_1(x) + \ldots + w_{m-1} \times \rho_{m-1}(x) + \left(1 - \sum_{i=1}^{m-1} w_i\right) \times \rho_m(x)$.

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.

3.1.1 An Implementation of the `Adjust` *Operation*

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

`Adjust(`$[S_1, .., S_k], e=0$`)`
1 `pick` j `such that` $[\![e]\!]S_j \neq [\![e]\!]S_k$.
2 `pick` $q \in \mathbb{Q}$ `such that` $q \neq 0$ `and` $q \neq [\![e]\!]S_i$ `for all` $i \in \{1, \ldots, k\}$.
3 `let` $\rho_0 = S_j \oplus_w S_k$, `where` $w = \frac{q-[\![e]\!]S_k}{[\![e]\!]S_j - [\![e]\!]S_k}$.
4 `for` i `= 1 to` $k-1$:
5 `let` S'_i `be the point at 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` $w_i = \frac{q}{q-[\![e]\!]S_i}$.
6 `return` $[S'_1, .., S'_{k-1}]$.

There are a few details in the definition of the `Adjust` procedure that deserve discussion. Line 1 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 2 is a linear time operation and the point ρ_0 is computed such that $[\![e]\!]\rho_0 = q$. Since we choose q such that ρ_0 and S_i are at different distances from the hyperplane $e=0$, the line joining ρ_0 and S_i intersects the hyperplane $e=0$ (in exactly one point). An example of the `Adjust` procedure is shown in Figure 2. The

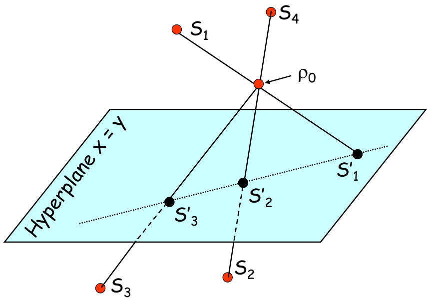

Fig. 2. An example of the `Adjust` procedure on a 4-point sample S, which satisfies the equality $z = x + y$. The adjustment is performed with respect to the equality $x = y$. The adjusted points S'_i are obtained as the intersections of the lines connecting the original points S_i with the point ρ_0.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

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

Proposition 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 Proposition 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 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.

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.

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

The loop starting in line 3 adjusts the sample incrementally so that it satisfies each equality literal in turn. Finally, the loop starting in line 6 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 3 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 for each $j = 1, \ldots, m$, the formula $\{t_i = 0\}_{i=1}^{k} \cup \{t'_j \neq 0\}$ is satisfiable. The loop starting in line 6 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 2 *Given a formula ϕ and a sample S, we say that S is* consistent *with ϕ if*

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

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.

Proposition 3 *If S is consistent with the formula $\phi \cup \{e = 0\}$, then* `Adjust`$(S, e = 0)$ *is consistent with the same formula.*

PROOF. Assume that S is consistent with $\phi \cup \{e = 0\}$. Let $S' =$ `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 Proposition 3, we can easily prove the following lemma:

Lemma 4 *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 Proposition 3 that the sample S in procedure `IsSatisfiable` always remains consistent with ϕ. Now, consider the following two cases.

- Suppose `IsSatisfiable` returns `false` in line 4. Then $S \models t_i + c = 0$. Since S is consistent with ϕ and $\phi \cup \{t_i + c = 0\}$ is unsatisfiable, it must be that ϕ is unsatisfiable.
- Suppose `IsSatisfiable` returns `false` in line 7. 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, then 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 5 (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 $k' \leq k$ is the maximum number of linearly independent equality literals in ϕ.*

This lemma along with Lemma 4 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 5 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 "loses" one assignment.

4 Extension to Uninterpreted Function Symbols

The theory of uninterpreted functions has one congruence axiom: $\left(\bigwedge_{i=1}^{n} e_i = e'_i\right) \Rightarrow f(e_1, .., e_n) = f(e'_1, .., e'_n)$ for any n-ary uninterpreted function f. We can efficiently detect the equivalences of expressions e_i and e'_i by comparing the values of these expressions under the random variable assignments. This allows us to reason about the congruence axiom of uninterpreted functions in an interesting manner. 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: $V(t_1+t_2) = V(t_1)+V(t_2)$, $V(t_1-t_2) = V(t_1)-V(t_2)$, $V(q\times t) = q \times V(t)$, $V(q) = q$, $V(f(t_1, \ldots, t_k)) = v_{f(t_1,..,t_k)}$. 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 $f(x+3)$, $f(z)$ 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`$'(\phi, R)$
  `let` $\mathcal{C}(\phi)$ `be` $\{t_i = 0\}_{i=1}^{k} \cup \{t'_i \neq 0\}_{i=1}^{m} \cup \{(\bigwedge_{j=1,..,k_i} s_{i,j} = s'_{i,j}) \Rightarrow v_i = v'_i\}_{i=1}^{\ell}$
  $S \leftarrow R$
  `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`

The `IsSatisfiable`$'$ procedure is similar to `IsSatisfiable` procedure with respect to processing of equalities $t_i = 0$ and disequalities $t'_i \neq 0$. Conditional equalities $(\bigwedge_{j=1,..,k_i} s_{i,j} = s'_{i,j}) \Rightarrow v_i = v'_i$ are handled by processing the equality $v_i = v'_i$ when $\bigwedge_{j=1,..,k_i} s_{i,j} = s_{i,j}$ is discovered to be true.

Note that `IsSatisfiable`$'(\phi, R)$ 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`$'(\phi, R)$ returns `false`. It also follows from Theorem 2 that if ϕ is satisfiable, then `IsSatisfiable`$'(\phi, R)$ 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 in the assignment 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 [2], in which non-convex theories are handled using backtracking. Similarly, a Shostak theorem prover [3] 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 requires some additional space. 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 different and interesting technique that has a slightly better space usage 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 simply adding this 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, 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.

5.1.1 *An Implementation of the* `Adjust′` *Operation*

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

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

The precondition for `Adjust′` ensures that an appropriate j can be found in line 2. It is a simple exercise to verify that $(S', \rho) = \texttt{Adjust}'(S, e)$ 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

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

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.

The following lemma states 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.

Lemma 6 (The Adjust-UnAdjust Lemma) *Let* $(S_1, \rho) = \mathtt{Adjust}'(S_0, e = 0)$ *and* S_2 *a sample that satisfies the same linear equalities as* S_1*, and* $S_3 = \mathtt{UnAdjust}(S_2, \rho)$*. 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, $S_0 \models t = 0$.

6 Producing Proofs

In this section, we show how to produce a proof for the correctness of the output of the algorithm. Such a procedure has several advantages. First of all, it can be used to convert the `IsSatisfiable`′ procedure, which is a Monte Carlo algorithm [2], to a Las Vegas algorithm [3] [4]. This can be done by repeating the algorithm on the same input until a proof can be produced. Secondly, it can be used as a mechanism to certify the output of the satisfiability procedure. This is useful for testing the implementation of the satisfiability procedure and in the context of proof-carrying code [5].

Proof production and its validity has also been considered in the context of the CVC (Cooperating Validity Checker) decision procedure [6,7].

[2] A Monte Carlo algorithm runs for a fixed number of steps for each input and produces an answer that is correct with a bounded probability.
[3] A Las Vegas algorithm always produces the correct answer, but its runtime for each input is a random variable whose expectation is bounded

We first discuss how to produce a proof when the `IsSatisfiable`$'$ procedure returns true on some input formula ϕ. Any point that satisfies ϕ is a certificate for the satisfiability of ϕ. We describe how to obtain one such point from the resulting sample at the end of the `IsSatisfiable`$'$ procedure.

Let $\mathcal{C}(\phi)$ be $E \cup \{t_i' \neq 0\}_{i=1}^m$ where E consists only of equality literals (possibly including some conditional equality literals). The resulting sample at the end of the `IsSatisfiable`$'$ procedure contains a point ρ_i for each $i \in \{1, .., m\}$ such that ρ_i satisfies the formula $\phi_i = E \cup \{t_i' \neq 0\}$. Clearly, such a point ρ_i is a certificate for the satisfiability of the formula ϕ_i. It follows from linear algebra that the formula ϕ is satisfiable if and only if all of the formulas ϕ_i are satisfiable. Hence, the collection of the points $\{\rho_i\}_{i=1}^m$ act as a certificate for the satisfiability of the formula ϕ.

It is also possible to produce a more succinct certificate, namely a single point ρ that satisfies the entire formula ϕ. We now describe a randomized construction to obtain such a point ρ. Let $\rho = \rho_1 \oplus_{w_1} \ldots \oplus_{w_{m-1}} \rho_m$ where $w_1, \ldots, w_{m-1}$ are weights chosen independently and u.a.r. from the set F. It is easy to show that ρ satisfies all the equalities that are satisfied by all of $\rho_1, \ldots, \rho_m$. Also, it is not hard to show that the probability that ρ satisfies any equality that is not satisfied by at least one of the points $\rho_1, \ldots, \rho_m$ is at most $\frac{1}{|F|}$. We now use these properties about ρ to obtain an upper bound on the probability that ρ does not satisfy $\mathcal{C}(\phi)$.

Let $e_1 = e_2 \Rightarrow e_3 = e_4$ be any conditional equality literal in E. If ρ does not satisfy $e_1 = e_2 \Rightarrow e_3 = e_4$, then it must be the case that ρ satisfies $e_1 = e_2$ and some ρ_j does not satisfy $e_1 = e_2$. (This is because if all ρ_j's satisfy $e_1 = e_2$, then all ρ_j's satisfy $e_3 = e_4$ and hence ρ also satisfies $e_3 = e_4$, thereby satisfying $e_1 = e_2 \Rightarrow e_3 = e_4$.) The probability that this happens is at most $\frac{1}{|F|}$. Let $t_j' \neq 0$ be some disequality literal. Note that ρ_j does not satisfy the equality $t_j' = 0$. Hence, the probability that ρ satisfies the equality $t_j' = 0$ is at most $\frac{1}{|F|}$. Hence, the probability that ρ does not satisfy the formula $\mathcal{C}(\phi)$ is at most $\frac{c+m}{|F|}$, where c is the number of conditional equality literals in E. This suggests that if F is big enough, then it is very likely that ρ will satisfy the formula $\mathcal{C}(\phi)$. If ρ does not satisfy $\mathcal{C}(\phi)$, then we can repeat the above construction until we obtain a point that does satisfy $\mathcal{C}(\phi)$.

6.2 Proof for UnSatisfiability of a Formula

In this section, we describe a procedure to construct a proof when the $\texttt{IsSatisfiable}'$ procedure returns false on some input formula ϕ. Note that there is a small probability that the formula ϕ is satisfiable even when the $\texttt{IsSatisfiable}'$ procedure returns false. Hence, the certificate that our procedure will construct will be valid only when the $\texttt{IsSatisfiable}'$ procedure returned a correct answer. However, it is easy to verify the validity of a certificate. If the certificate is not valid, then the $\texttt{IsSatisfiable}'$ procedure can be repeated on the same input formula ϕ until we obtain a valid certificate, or the $\texttt{IsSatisfiable}'$ procedure returns true. We prove that the probability that a valid certificate will be produced, given that the $\texttt{IsSatisfiable}'$ procedure returns false, is high. Thus, the expected value of the number of repetitions required is small.

A certificate for a valid implication $\left(\bigwedge_{i=1}^{k} t_i = 0\right) \Rightarrow (t = 0)$ is a sequence of constants $\lambda_1, \ldots, \lambda_k$ such that $t = \sum_{i=1}^{k} \lambda_i t_i$. We can produce such a certificate if, for each $i \in L$, we have a point ρ_i that satisfies the equality literals $t_j = 0$ for all $1 \leq j \leq i-1$, but does not satisfy $t_i = 0$, where L is the following set.

$$L = \{i \mid 1 \leq i \leq k, \bigwedge_{j=1}^{i-1} t_j = 0 \not\Rightarrow t_i = 0\}$$

Note that $\{t_i\}_{i \in L}$ is a maximal set of linearly independent t_i's. It thus follows from linear algebra that there exist $\{\lambda_i\}_{i \in L}$ such that $t = \sum_{i \in L} \lambda_i t_i$. Thus, for any $i \in L$, $[\![t]\!]\rho_i = \sum_{j \in L} \lambda_j \times [\![t_j]\!]\rho_i = \sum_{j \in L, j \geq i} \lambda_j \times [\![t_j]\!]\rho_i$, or equivalently, $\lambda_i = \frac{[\![t]\!]\rho_i - \sum_{j \in L, j > i} \lambda_j \times [\![t_j]\!]\rho_i}{[\![t_i]\!]\rho_i}$. Hence, given these points, we can compute the certificate coefficients as follows.

> **for** i = k **downto** 1:
>
>   **if** $i \in L$, **then** $\lambda_i = \frac{[\![t]\!]\rho_i - \sum_{j \in L, j > i} \lambda_j \times [\![t_j]\!]\rho_i}{[\![t_i]\!]\rho_i}$
>
>   **else** $\lambda_i = 0$

We now describe how to certify the result of $\texttt{IsSatisfiable}'$ procedure when it returns false on some input formula ϕ. Let $\mathcal{C}(\phi)$ be $\{t_i = 0\}_{i=1}^{k} \cup \{t'_i \neq 0\}_{i=1}^{m} \cup \{(\bigwedge_{j=1,..,k_i} s_{i,j} = s'_{i,j}) \Rightarrow v_i = v'_i\}_{i=1}^{\ell}$. Let e be the term defined as follows. If $\texttt{IsSatisfiable}'$ returns false in $\texttt{Assume}$ procedure for some input equality $t = 0$, then e is equal to t. Else if $\texttt{IsSatisfiable}'$ returns false in line 10 for some i, then let e be equal to t'_i. Let d be the number of successful calls made to the `Assume` procedure before the `IsSatisfiable'` procedure returns false. Let $(S^i, e_i = 0)$ be the input to the `Assume` procedure in the i^{th} call. For any $i \in \{1, \ldots, d\}$, if the i^{th} call to the `Assume` procedure is made in line 8, then let E_i be the collection of the corresponding equalities $\{s_{w,j} - s'_{w,j} = 0\}_{j=1}^{k_w}$ in the guard in line 8. We assume that the `Assume` procedure uses the `Adjust'` function described in Section 5.1.1 instead of the `Adjust` function. Let L be the set of all i such that the i^{th} call to the `Assume` procedure results in a call to the `Adjust'` procedure, i.e. $S^i \not\models e_i = 0$. For any $i \in L$, let ρ_i be the point returned by the `Adjust'` procedure. Note that ρ_i satisfies the equality literals $e_j = 0$ for all $1 \leq j \leq i-1$ but does not satisfy $e_i = 0$.

A proof for the result of `IsSatisfiable'` procedure consists of a proof of the implication $\left(\bigwedge_{i=1}^{d} e_i = 0\right) \Rightarrow e = 0$, and a proof of the implication $\left(\bigwedge_{j=1}^{i-1} e_j = 0\right) \Rightarrow t = 0$ for each $t = 0 \in E_i$. A certificate for each of these implications can be generated by the procedure described above using the points $\{\rho_i\}_{i \in L}$, provided for all $i \notin L$, $\bigwedge_{j=1}^{i-1} e_j = 0 \Rightarrow e_i = 0$. If these certificates are valid, then we have a valid proof for the unsatisfiability of the formula. Else we can repeat the `IsSatisfiable'` procedure until we obtain a valid certificate, or the `IsSatisfiable'` procedure returns true.

We now show that the probability of obtaining an invalid certificate is small (given that the `IsSatisfiable'` procedure returns false). Hence, the expected number of repetitions of `IsSatisfiable'` procedure until a proof can be produced is also small. The certificate would be valid if for all $i \notin L$, $\bigwedge_{j=1}^{i-1} e_j = 0 \Rightarrow e_i = 0$. The probability that $S^i \models e_i = 0$ when $\bigwedge_{j=1}^{i-1} e_j = 0 \not\Rightarrow e_i = 0$ is the same as the probability that the initial random sample R is inconsistent with the formula $\{e_j = 0\}_{j=1}^{i-1} \cup \{e_i \neq 0\}$. It thus follows from Lemma 9 that for any $i \notin L$, the probability that $S^i \models e_i = 0$ when $\bigwedge_{j=1}^{i-1} e_j = 0 \not\Rightarrow e_i = 0$ is bounded above by $\tau = \frac{|F|}{|F|-3r}\left(\frac{3r}{|F|}\right)^{r-k'}$, where r is the number of points in the initial random sample R, and k' is the maximum number of linearly independent equality literals in $\mathcal{A}(\phi)$. Hence, the probability of obtaining an invalid certificate is bounded above by $\tau k'$. This probability can be made arbitrarily small by increasing the value of $|F|$ or r.

7 Optimization: Conversion of terms to terms with only a single unary function symbol

In this section, we describe a randomized transformation that converts terms in a formula involving (possibly multiple) uninterpreted function symbols of any finite arity to terms involving only one unary uninterpreted function symbol. Such a transformation can be used to make the implementation simpler and possibly efficient. This is another example of obtaining simplicity at the expense of making soundness probabilistic.

Let V be any injective mapping from uninterpreted function symbols f to rationals. Let d be the maximum arity of any uninterpreted function symbol f. Let $w_0, .., w_d$ be some rationals chosen independently and u.a.r. from some finite subset F of $\mathbb{Q}$. Let g be a new unary uninterpreted function symbol. The following transformation T transforms any term to a term that involves only one uninterpreted function symbol, namely g.

$$\begin{aligned}
T(x) &= x \\
T(q) &= q \\
T(t_1 + t_2) &= T(t_1) + T(t_2) \\
T(t_1 - t_2) &= T(t_1) - T(t_2) \\
T(q \times t) &= q \times T(t) \\
T(f(t_1, .., t_k)) &= g(w_0 V(f) + \sum_{i=1}^{k} w_i T(t_i))
\end{aligned}$$

7.1 Correctness

We now prove the correctness of the transformation T. We say that the transformation T is correct for a formula ϕ if any two subterms that occur in ϕ are equal iff they are equal after the transformation. For any formula $\phi = \{t_i = 0\}_{i=1}^{n} \cup \{t'_i \neq 0\}_{i=1}^{m}$, let $T(\phi)$ denote the formula $\{T(t_i) = 0\}_{i=1}^{n} \cup \{T(t'_i) \neq 0\}_{i=1}^{m}$. Note that if the transformation T is correct for a formula ϕ, then ϕ is satisfiable iff the formula $T(\phi)$ is satisfiable. The following theorem states that the transformation T is correct for a formula ϕ with high probability over the choice of the random weights $\{w_i\}_{i=0}^{d}$.

Theorem 3 *For any two subterms t and t' in a formula ϕ, $t = t'$ iff $T(t) = T(t')$, with high probability over the choice of the random weights $\{w_i\}_{i=0}^{d}$.*

The proof of Theorem 3 follows easily from Lemma 7 and Lemma 8 stated and proved below. Lemma 7 states that any two equal subterms of formula

ϕ are also equal after the transformation T. Lemma 8 implies that any two unequal subterms of formula ϕ are not equal after the transformation T with high probability.

Lemma 7 (Completeness) *For any two subterms t and t' in formula ϕ, if $t = t'$, then $T(t) = T(t')$.*

Lemma 7 can be proved easily by induction on size of the terms.

Before stating the soundness lemma, we introduce some notation. Let the height $H(t)$ of a term t be defined as follows.

$$
\begin{aligned}
H(x) &= 1 \\
H(q) &= 1 \\
H(t_1 + t_2) &= max\{H(t_1), H(t_2)\} \\
H(t_1 - t_2) &= max\{H(t_1), H(t_2)\} \\
H(q \times t) &= H(t) \\
H(F(t_1, .., t_k)) &= 1 + max\{H(t_1), .., H(t_k)\}
\end{aligned}
$$

Lemma 8 (Soundness) *Let n_h be the number of pairs of function subterms of height less than or equal to h in formula ϕ. Let E_h be the event that there exist two subterms t and t' of height less than or equal to h in formula ϕ such that $t \neq t'$ and $T(t) = T(t')$. The probability (over the choice of the random weights $\{w_i\}_{i=0}^{d}$) that event E_h occurs is at most $\frac{n_h}{|F|}$.*

PROOF. The proof is by induction on the height h of the subterms that occur in formula ϕ. We prove the inductive case. (Proof for the base case is similar). We assume that $\Pr(E_h) \leq \frac{n_h}{|F|}$ and prove that $\Pr(E_{h+1}) \leq \frac{n_{h+1}}{|F|}$.

Let E'_h be the event that there exist two *function* subterms t and t' of height less than or equal to h such that $t \neq t'$ and $T(t) = T(t')$. It is not difficult to see that the event E_h occurs iff the event E'_h occurs. Hence, it suffices to prove that $\Pr(E'_{h+1}) \leq \frac{n_{h+1}}{|F|}$.

Let $t = f(a_1, .., a_k)$ and $t' = f'(b_1, .., b_{k'})$ be two function subterms in formula ϕ such that the height of any one of them, say t, is at least $h + 1$ and $t \neq t'$. We show that the probability that $T(t) = T(t')$ is at most $\frac{1}{|F|}$. At least one of the following cases arise.

- $f \neq f'$. Note that
  $T(t) = T(t')$
  $\iff w_0 V(f) + \sum_{i=1}^{k} w_i T(a_i) = w_0 V(f') + \sum_{i=1}^{k'} w_i T(b_i)$

$\iff w_0 = \frac{c}{d}$, *where* $d = V(f) - V(f')$, *and* $c = \sum_{i=1}^{k'} w_i b_i - \sum_{i=1}^{k} w_i a_i$

Note that $d \neq 0$, since $f \neq f'$ and the mapping V is injective. If $\frac{c}{d}$ is not a constant, then $w_0 \neq \frac{c}{d}$. If $\frac{c}{d}$ is a constant, then the probability that $w_0 = \frac{c}{d}$ is $\frac{1}{|F|}$. Hence, the probability that $w_0 = \frac{c}{d}$ is at most $\frac{1}{|F|}$.

- $f = f'$ (and thus $k = k'$), but $a_j \neq b_j$ for some $j \in \{1, .., k\}$. Note that $T(t) = T(t')$

  $\iff w_0 V(f) + \sum_{i=1}^{k} w_i T(a_i) = w_0 V(f') + \sum_{i=1}^{k'} w_i T(b_i)$

  $\iff w_j = \frac{c}{d}$, where $d = T(a_j) - T(b_j)$, and $c = \sum_{i=1}^{j-1} w_i (T(b_i) - T(a_i)) + \sum_{i=j+1}^{k} w_i (T(b_i) - T(a_i))$.

  Note that $d \neq 0$ (under the assumption that the event E_h does not occur). The probability that $w_0 = \frac{c}{d}$ is at most $\frac{1}{|F|}$.

The number of pairs of function subterms in ϕ such that at least one of them has height $h+1$ is $n_{h+1} - n_h$. Thus, $\Pr(E'_{h+1} \mid \neg E_h) \leq (n_{h+1} - n_h)\frac{1}{|F|}$. Hence,

$$\begin{aligned}
\Pr(E_{h+1}) &= \Pr(E'_{h+1}) \\
&\leq \Pr(E_h) + \Pr(E'_{h+1} \mid \neg E_h) \\
&\leq \frac{n_h}{|F|} + \frac{n_{h+1} - n_h}{|F|} \\
&= \frac{n_{h+1}}{|F|}
\end{aligned}$$

Lemma 8 suggests that the error probability in the transformation T can be made arbitrarily small by choosing random weights $\{w_i\}_{i=1}^{d}$ from a large enough F.

8 Experimental Results

We have implemented the `IsSatisfiable`′ procedure (described in Section 4) in programming language 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 [4]. Our implementation also uses the optimization described in Section 7. We did not observe a significant difference in performance based on this optimization; however this optimization made the implementation simpler.

We compared the running-time performance of our implementation with the SRI's `ICS` (version 1.0) and Stanford's `SVC` and `CVC Lite` (version 20040606 - the latest build version, as recommended by the CVC Lite developers) decision procedure packages. `ICS` (Integrated Canonized and Solver) is implemented in Ocaml [8] and is based on the refinement of Shostak's algorithm by Ruess and Shankar [9]. It can decide a fragment of first-order logic where the terms are built from uninterpreted function symbols and operators from a combination of datatypes including arithmetic, functional arrays, tuples, cotuples, and fixed-sized bitvectors. `ICS` uses arbitrary precision rational numbers from the GNU multi-precision library (GMP). `SVC` (Stanford Validity Checker) is implemented in C++, and is used to check the validity of quantifier-free first-order formulas over several theories including real linear arithmetic, arrays, uninterpreted functions [10]. It has been used primarily for the formal verification of hardware designs. `CVC Lite` Lite (Cooperating Validity Checker Lite) is a successor of `SVC` [11]. It provides a C++ library with a well-defined API that provides support for the theory of linear arithmetic over reals as well as integers, and their combination. It can also handle a limited form of non-linear arithmetic. It supports arbitrary precision arithmetic using GMP.

Figure 3 shows the time in milliseconds taken by our implementation and the time in seconds taken by `ICS`, `SVC` and `CVC Lite` while deciding validity of several formulas that involve only linear arithmetic. Column `Rand` shows the time taken by our implementation when run with the best possible parameters. This includes performing arithmetic operations over a small field (in this case $Z_{268435399}$, so that the arithmetic can be performed using 32-bit integers) and working with as few points (in the initial random sample) as required. In particular, we chose 5 more points than the number of `Adjust` operations performed; this was sufficient to guarantee a very low error probability. The number of `Adjust` operations is equal to the number of *independent* equality literals in the formula. Since this cannot be determined beforehand, we chose the number of equality literals as a good upper approximation to the number of `Adjust` operations for the purpose of determining the number of points to use. The experiments were performed on a 1.7 GHz Pentium 4 machine running Linux 2.4.5. The formulas in the table have been classified based on the number of equality literals in the formula and the maximum number of variables that occur in each equality literal. Our implementation is 2 to 4 orders of magnitude faster than the other tools. This is primarily due to the fact that our algorithm performs arithmetic over a small randomly chosen field and hence all arithmetic operations can be performed efficiently. The

# Equalities	Sparsity	Rand (ms)	ICS (sec)	SVC (sec)	CVC Lite (sec)
20	5	0.30	0.02	0.01	0.05
20	10	0.37	0.08	0.15	0.05
20	20	0.83	0.37	0.18	1.83
20	40	1.38	0.50	2.84	mem
20	60	1.63	0.55	6.01	mem
20	80	2.24	0.58	6.31	mem
20	100	2.40	0.59	7.08	mem
40	3	0.87	0.04	0.21	0.05
40	5	1.62	0.08	0.51	1.83
40	10	2.24	1.09	8.62	mem
40	20	4.54	4.24	32.91	mem
40	40	6.75	6.01	67.65	mem
40	50	7.23	6.20	time	mem
60	2	1.58	0.04	0.52	0.24
60	4	2.71	0.68	2.12	mem
60	8	6.21	10.24	26.86	mem
60	15	13.2	20.93	time	mem
60	30	17.24	25.64	time	mem
60	50	22.6	27.76	time	mem
80	1	0.76	0.02	0.5	0.18
80	2	2.71	0.11	1.60	1.57
80	4	7.39	2.59	15.96	mem
80	8	13.1	39.96	time	mem
80	15	27.6	67.28	time	mem
80	30	43.4	81.5	time	mem
80	50	50.5	86.32	time	mem
100	1	1.26	0.05	1.84	0.33
100	2	5.32	0.64	5.25	mem
100	4	17.59	29.33	54.00	mem
100	6	23.49	66.19	time	mem
100	8	42.00	time	time	mem
100	10	40.8	time	time	mem
125	1	4.50	0.12	3.56	0.68
125	2	11.9	1.06	17.38	mem
125	3	26.8	38.73	time	mem
125	4	33.33	time	time	mem
150	1	8.21	0.19	9.4	1.92
150	2	28.57	10.69	71.63	mem
150	3	56.57	time	time	mem
150	4	80.00	time	time	mem
200	1	27.78	1.55	40.90	mem
200	2	83.33	377.68	time	mem

Table 1
This table compares the time taken by our implementation `Rand` (in milliseconds), `ICS` (in seconds), `CVC Lite` (in seconds) and `SVC` (in seconds) on several example formulas involving only linear arithmetic. The number of points used by `Rand` were 5 more than the number of adjust operations performed by it. Column `# Equalities` denotes the number of equality literals. The number of variables in each formula is twice the number of equality literals in that formula. Column `sparsity` denotes the maximum number of variables in each equality literal as a percentage of the total number of variables in the formula. `time` denotes "more than 100 seconds", and `mem` denotes "out of memory" on a machine with 2.3 GB of virtual memory.

`# Equalities`	`Depth`	`Rand` (sec)	`ICS` (sec)	`SVC` (sec)	`CVC Lite` (sec)
20	1	0.001	0.02	0.09	0.09
20	2	0.007	0.04	0.09	mem
20	3	0.016	0.08	0.19	mem
20	4	0.11	0.13	0.12	mem
40	1	0.007	0.04	0.22	0.16
40	2	0.01	0.09	0.21	48.92
40	3	0.06	0.19	0.32	mem
40	4	1.97	0.36	0.25	mem
60	1	0.008	0.05	0.24	0.17
60	2	0.04	0.44	1.55	mem
60	3	0.2	0.31	0.53	mem
60	4	2.27	0.84	0.85	mem
80	1	0.009	0.06	0.24	0.25
80	2	0.04	0.19	0.64	mem
80	3	0.32	0.58	1.61	mem
80	4	1.91	1.17	1.29	mem
100	1	0.01	0.08	0.27	0.32
100	2	0.03	0.16	0.44	14.88
100	3	0.08	0.28	0.41	mem
100	4	3.16	0.55	0.54	mem
125	1	0.02	0.11	0.39	0.47
125	2	0.03	0.20	1.64	mem
125	3	0.38	0.61	0.92	mem
125	4	11.61	1.80	1.17	mem
150	1	0.02	0.12	0.38	0.56
150	2	0.05	0.23	1.96	mem
150	3	0.24	0.69	2.36	mem
150	4	2.36	0.71	1.32	mem
200	1	0.04	0.16	0.54	0.8
200	2	0.09	0.27	2.74	9.04
200	3	0.44	2.11	17.86	mem
200	4	7.67	2.40	6.41	mem
300	1	0.11	0.27	1.14	sf
300	2	0.18	0.47	4.12	sf
300	3	2.37	2.07	7.93	sf
300	4	28.63	7.95	15.67	sf
300	5	13.43	6.23	22.15	sf
400	1	0.28	0.39	1.93	sf
400	2	0.36	0.51	3.4	sf
400	3	7.94	12.07	26.75	sf
400	4	48.84	12.28	24.07	sf
400	5	time	18.94	26.68	sf
500	1	0.43	0.47	2.34	sf
500	2	0.75	0.85	12.08	sf
500	3	5.1	10.3	50.45	sf
500	4	12.13	39.01	time	sf
500	5	time	57.56	30.47	sf

Table 2
This table compares the time (in seconds) taken by our implementation `Rand`, `ICS`, `CVC Lite` and `SVC` on several examples that involve combination of linear arithmetic with uninterpreted functions. The number of points used by `Rand` were 5 more than the number of adjust operations performed by it. `#Equalities` denotes the number of equality literals. `Depth` denotes the maximum switching depth of the two theories in the literals. `time` denotes “more than 100 seconds”, `mem` denotes “out of memory” on a machine with 2.3 GB of virtual memory, and `sf` denotes “segmentation fault”.

deterministic algorithms implemented by other tools use arbitrary precision arithmetic, which is expensive. Note that the speedup of our algorithm is more when the number of variables in each equality literal is more. This is because greater number of variables in each literal lead to large intermediate arithmetic constants while manipulating those literals symbolically.

Figure 4 shows the time in seconds taken by our implementation `Rand`, `ICS`, `SVC` and `CVC Lite` while deciding validity of formulas that involve both linear arithmetic and uninterpreted functions. Here also, `Rand` performed arithmetic over the field $Z_{268435399}$ and worked with points that were 5 more than the number of `Adjust` operations performed. However, in this case, the number of equality literals do not provide an upper bound on the number of `Adjust` operations. Hence, ideally, the tool needs to start with a small number of points, and then increase the number of points on demand. However, for quick prototyping, we have not yet implemented this feature in the tool. For experiments, we first ran our tool with large enough points to determine the number of `Adjust` operations performed, and then ran it again and timed it with 5 more points than the number of `Adjust` operations. We suspect that because of this methodology of quick prototyping, the time measurement of `Rand` is slightly better than otherwise. The formulas in the table have been classified based on the number of equality literals in the formula and the maximum number of theory switches in the literals. The speed-up of our implementation over other tools decreases with the increase in the switching depth of theories in the literals. This is because the cost of arithmetic operations gets dominated by the cost of sharing of equalities between the two theories. In fact, both `ICS` and `SVC` perform better than `Rand` when switching depth is 4 or more. This suggests that they handle the combination of uninterpreted functions with linear arithmetic in a more efficient manner than our implementation.

As expected, `Rand` never returned a false answer in these experiments.

9 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 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 small 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 [2] 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 [3] 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, $[[\![t]\!]S_1, .., [\![t]\!]S_r]$ 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 incomplete and potentially non-terminating. Recently, Ruess and Shankar [9] 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 [12] 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.

10 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 and the size of the set from which the random values are chosen. 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] G. Nelson, D. Oppen, Simplification by cooperating decision procedures, ACM Transactions on Programming Languages and Systems 1 (2) (1979) 245–257.

[3] R. E. Shostak, Deciding combinations of theories, Journal of the ACM 31 (1) (1984) 1–12.

[4] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.

[5] G. C. Necula, Proof-carrying code, in: Proceedings of the 24th ACM Symposium on Principles of Programming Languages, 1997, pp. 106–119.

[6] A. Stump, Checking validities and proofs with cvc and flea, Ph.D. thesis, Stanford University, available from http://www.cse.wustl.edu/ stump (2002).

[7] A. Stump, D. L. Dill, Generating proofs from a decision procedure, in: A. Pnueli, P. Traverso (Eds.), Proceedings of the FLoC Workshop on Run-Time Result Verification, Trento, Italy, 1999.

[8] J.-C. Filliâtre, S. Owre, H. Rueß, N. Shankar, ICS: Integrated Canonization and Solving, in: G. Berry, H. Comon, A. Finkel (Eds.), Computer-Aided Verification, CAV '2001, Vol. 2102 of Lecture Notes in Computer Science, Springer-Verlag, Paris, France, 2001, pp. 246–249.

[9] H. Ruess, N. Shankar, Deconstructing Shostak, in: 16th Annual IEEE Symposium on Logic in Computer Science (LICS '01), IEEE, Washington - Brussels - Tokyo, 2001, pp. 19–28.

[10] C. W. Barrett, D. L. Dill, L. Levitt, Validity checking for combinations of theories with equality, in: M. K. Srivas, A. Camilleri (Eds.), Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (Palo Alto, CA), Vol. 1166 of Lecture Notes in Computer Science, Springer, 1996, pp. 187–201.

[11] C. Barrett, S. Berezin, Cvc lite: A new implementation of the cooperating validity checker, in: R. Alur, D. Peled (Eds.), Proceedings of the 16th International Conference on Computer Aided Verification, CAV'04 (Boston, Massachusetts), Lecture Notes in Computer Science, Springer, 2004, pp. 515–518.

[12] S. Gulwani, G. C. Necula, Discovering affine equalities using random interpretation, in: The 30th Annual ACM Symposium on Principles of Programming Languages, ACM, 2003, pp. 74–84.

A Proof of Consistent Random Sample Lemma

Lemma 5 (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 9 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 9 *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,*
$\Pr_R[\exists \alpha_1, .., \alpha_{k'}$ *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 $z_1, .., z_{k'}$:

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

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 [4] 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 $\{\mathcal{E}_I \mid I \subseteq \{1, .., r\}, 1 \in I, |I| \leq k'\}$ 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,

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

It now follows from the claim stated and proved below that

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

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

Here is some intuition behind Inequality A.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 Inequalities (A.1) and (A.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 10 *For any subset I of $\{1, .., r\}$, $\Pr_R[\mathcal{E} \cap \mathcal{E}_I] \leq \left(\frac{1}{|F|}\right)^{r-|I|}$.*

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, $\Pr_R[\mathcal{E}_I] = \Pr_R[\bigwedge_{i \in I} \mathcal{F}_{I,i} \ \wedge \bigwedge_{i \in \{1,..,r\}-I} \mathcal{G}_{I,i}]$.

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 9 that

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

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,

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

The required result now follows from Inequalities (A.3) and (A.4).