Discovering Affine Equalities Using Random Interpretation

Sumit Gulwani George C. Necula
University of California, Berkeley
{gulwani,necula}@cs.berkeley.edu

ABSTRACT

We present a new polynomial-time randomized algorithm for discovering affine equalities involving variables in a program. The key idea of the algorithm is to execute a code fragment on a few random inputs, but in such a way that all paths are covered on each run. This makes it possible to rule out invalid relationships even with very few runs.

The algorithm is based on two main techniques. First, both branches of a conditional are executed on each run and at joint points we perform an affine combination of the joining states. Secondly, in the branches of an equality conditional we adjust the data values on the fly to reflect the truth value of the guarding boolean expression. This increases the number of affine equalities that the analysis discovers.

The algorithm is simpler to implement than alternative deterministic versions, has better computational complexity, and has an extremely small probability of error for even a small number of runs. This algorithm is an example of how randomization can provide a trade-off between the cost and complexity of program analysis, and a small probability of unsoundness.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Program analysis*

General Terms

Algorithms, Theory, Verification

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, Air Force contract no. F33615-00-C-1693, 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.

POPL'03, January 15–17, 2003, New Orleans, Louisiana, USA.

Keywords

Affine Relationships, Linear Equalities, Random Interpretation, Randomized Algorithm

1. INTRODUCTION

In this paper, we take a fresh look at program analysis and explore what can be learned about a program by running it on a small number of input values. At first sight, the answer appears to be discouraging. After all, we know that testing is only as good as the coverage of the test cases, and that even loop-free programs have an exponential number of paths.

The problem is that with a small number of inputs we cannot cover the entire program. This is because at each branching point we make a binary decision: if the decision variable has value 0 we take one path and if it has value 1 we take the other. But what if we relax the semantics of the program and take a "middle" path instead? For example, we could take 30% of one path and 70% of the other. This means that we execute both paths, and at the join point we combine the two values for each variable using a factor 30/70 (i.e., the value of variable x after the join is $0.3\times x_1 + 0.7\times x_2$ where x_1 and x_2 are the values on the joining branches). This gives us a continuum of choices at each branch, among which we choose randomly, with the overall effect that each such "run" of the program involves all of the paths.

It is not obvious that such a contrived execution has much in common with a real run of the program. However, we prove in this paper that this strategy captures most of the affine relationships among variables at any point in a program. An affine relationship among variables x_i $(i = 1\,..\,n)$ is a relationship of the form $\sum_{i=1}^{n} \alpha_i x_i + c = 0$, where α_i $(i = 1..n)$ and c are some constants. Several classical data flow analysis problems can be modeled as the problem of detecting affine relationships among variables. Examples are: constant propagation (such as $x = 2$), discovery of symbolic constants (such as $x = 5 \times N + 1$), detection of common sub-expressions. Several loop invariant computations and loop induction variables can also be identified by detecting affine relationships. Translation validation [11, 10] also requires checking the equivalence of variables in two versions of a program before and after optimization.

Consider, for example, the program shown in Figure 1 (ignoring for the moment the annotations shown on the side). Of the two assertions at the end of the program, the first is true on all four paths, and the second is true on three of them (it is false when the first conditional is false and the second is true). Regular testing would have to exercise that precise path to avoid inferring that the second equality

holds. Instead, we propose to use a non-standard execution model. At branches we proceed on both the true and false branch. At joins we choose a random weight w, and we use it to combine the values v_1 and v_2 of variable v in the two branches, as follows: $w \times v_1 + (1-w) \times v_2$. In the example, all variables are dead on entry, so we start the execution with some arbitrary values (shown as $*$ in the figure). We use the random weights $w_1 = 5$ for the first branch and $w_2 = -3$ for the second branch. We can then verify easily that the resulting state satisfies the first assertion but does not satisfy the second. Thus, in one run of the program we have noticed that one of the exponentially many paths breaks the invariant.

The price for the simplicity of this analysis is that sometimes it may be unsound, meaning that it may incorrectly claim that a relationship holds, even when there are execution paths on which the relationship does not hold. However, we prove that the probability that this happens can be made infinitesimally small, so that for practical purposes we can assume that the analysis is a sound one. A close analysis of our example shows that there are quite a few choices for w_1 and w_2 that would make it appear that the second assertion also holds (precisely those when either $w_1 = 1$ or $w_2 = 0$). If the random choice of weights were restricted to 0 or 1 (that is, those modeling actual executions in the program) then the probability of unsound results in one run would be $\frac{3}{4}$ in this case, or $\frac{2^n-1}{2^n}$ worst case in general for a program of size n. However, if we relax the choice of weights and let w_1 and w_2 be 32-bit numbers, there are $2^{33}-1$ choices for which we obtain incorrect results. Since the total number of choices for choosing w_1 and w_2 are 2^{64}, the probability of obtaining an unsound conclusion is less than 2^{-31}.

If we try to discover relationships by analyzing the values of the variables, we may draw incorrect conclusions. For example, in Figure 1, it may appear that $a = -4$ is an invariant at the end of the program, which is incorrect. To avoid this problem, we execute the program several times and then look for common relationships among all those executions. A close analysis of our example shows that if we execute the program once more, the probability of a evaluating to -4 again is precisely the probability that we choose the random weight w_1 to be 5 again, which is equal to 2^{-32}, if the weights are 32-bit numbers.

There are numerous algorithms in the literature [2, 5, 9] that can verify or discover affine relationships. Our proposed algorithm differs from these in several respects. It is simpler to implement since it resembles an interpreter, and does not involve complex computations of convex hulls [5] or affine unions of spaces [9]. Our algorithm can be wrong on rare occasions, but by repeated runs we can reduce the probability of failure to a negligible value. Finally, the inferred relationships are represented implicitly in the form of all the linear relationships that the set of runs satisfies. However, the information is easy to extract when needed, and we can quickly answer questions about whether a certain relationship holds by simply checking whether all runs satisfy that relationship.

In contrast to the simplicity of the algorithm, the formal proofs that the soundness probability is high are subtle, as is usually the case with randomized algorithms. In this paper we present the formal proofs for the case when the program involves only linear computations. We discuss first how the algorithm handles basic control-flow elements such

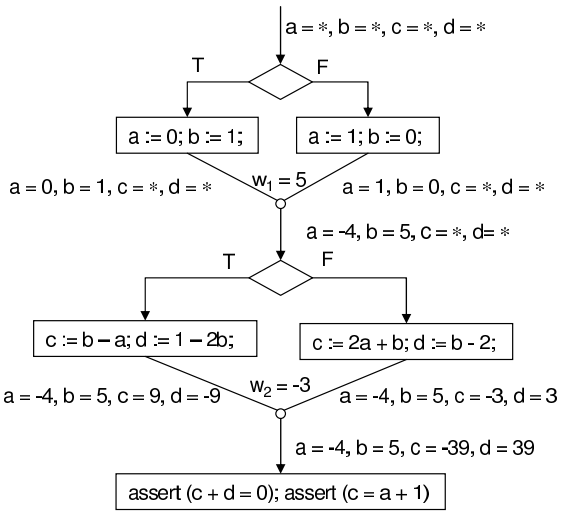

Figure 1: A code fragment with four paths. Of the two equations asserted at the end the first one holds on all paths but the second one holds only on three paths. The numbers shown next to each edge represent values in a random execution. The join points are affine combinations of the inputs to the join with the weights $w_1 = 5$ and respectively $w_2 = -3$.

as assignments (Section 3), joins (Section 4) and branches (Section 5). Then, in Section 6, we put these pieces together and we state and prove the main completeness theorem and the probabilistic soundness theorem. In Section 7 we make a number of observations related to possible extensions of these techniques beyond linear arithmetic.

2. NOTATION

In this paper we work with a simple imperative language containing the following affine expressions over integers $\mathbb{Z}$ (here $q \in \mathbb{Z}$ and x is one of n variables):

$$e ::= x \quad | \quad q \quad | \quad e_1 + e_2 \quad | \quad e_1 - e_2 \quad | \quad e \times q$$

We write $e[e'/x]$ to denote the result of substituting e' for x in e.

The random interpreter needs to perform divisions and hence it must work with values chosen from a field $\mathbb{F}$ (recall that a field is a mathematical structure whose elements have multiplicative inverses). For the purpose of the discussion in the following three sections, the reader may consider that the field $\mathbb{F}$ is the set of rationals $\mathbb{Q}$. However, as we will see later, another choice for $\mathbb{F}$ appears to be more appropriate both for technical as well as for implementation reasons.

A *state* ρ is an assignment of field values to the n variables. Occasionally, in order to expose the geometric intuitions behind the algorithms, we also refer to the n variables as *coordinates* and to states as *points* in $\mathbb{F}^n$. We write $[\![e]\!]\rho$ for the meaning of e in the state ρ (using the usual interpretation of the arithmetic operations over $\mathbb{F}$). The notation $\rho[x \leftarrow q]$ denotes the state obtained from ρ by setting the value of variable x to q. We say that a state ρ satisfies an equation $e = 0$ when $[\![e]\!]\rho = 0$. We write $\rho \models e = 0$ when this is the case.

Our algorithm requires several runs with random input

values. The random interpreter can be thought of as executing the program in parallel on a collection of states. We refer to such a collection of states S as a *sample* and we write S_i to refer to the ith element of the sample S (the state corresponding to the ith run). In the geometric interpretation, a sample is a sequence of points. Throughout this paper we assume that all samples have r elements, where r is a parameter of the algorithm. We say that a sample satisfies a linear equation $e = 0$ when all of its states satisfy the equation. We write $S \models e = 0$ when this is the case.

Whenever the interpreter must choose some field value at random, it does so independently of the previous choices and uniformly at random (u.a.r.) from some finite subset $\tilde{F}$ of $\mathbb{F}$ of size d, which is another parameter of the algorithm. With a larger value of d, the probability of errors in the algorithm is smaller, but the interpreter must have more random bits to make the choices.

Throughout the paper we use the *affine combination* operation on field values and on states. An affine combination is a weighted average of a number of values such that the sum of the weights is 1. We perform this operation most often on two values, in which case we write $q \oplus_w q'$ for the value $q \times w + q' \times (1 - w)$. We refer to w as the weight of the combination. We extend this operation to states, in which case we perform the combination with the same weight for each variable. If the states ρ_1 and ρ_2 are viewed as points in $\mathbb{F}^n$ then their affine combinations are the points situated on the line passing through ρ_1 and ρ_2. We further extend the affine combination to samples, in which case we combine each pair of corresponding states using a separate weight factor:

$$(S \oplus_{[w_1,\ldots,w_r]} S')_i = S_i \oplus_{w_i} S'_i \quad (i = 1\,..\,r)$$

In the following sections we consider separately the operation of the random interpreter on various nodes of a flowchart. Then, in Section 6 we put the pieces together and we define precisely the random interpreter algorithm.

3. THE ASSIGNMENT OPERATION

In the case of assignments the random interpreter behaves exactly as a concrete interpreter. For the assignment $x := e$, it transforms each state in the sample by setting x to the value of e in that state. If the sample before the assignment is S then the sample after the assignment is S' such that:

$$S'_i = S_i[x \leftarrow [\![e]\!]S_i]$$

4. THE UNION OPERATION

The random interpreter executes both branches of conditionals. Assume that the interpreter reaches a join point with two samples S and S'. Each of these samples encodes implicitly a number of affine relationships between variables. In order to continue with only one sample after the join point, we perform a union operation in which the resulting sample S^u encodes (implicitly) all of the relationships that S and S' have in common, and no other relationships. In previous work, the union operation is quite involved, requiring complex algorithms for computing the affine union of affine spaces [9] or convex hulls when affine inequalities are also handled [5]. In contrast, a random interpreter simply chooses r random weights $w_1, \ldots, w_r$ from $\tilde{F}$ and computes

$$S^u = S \oplus_{[w_1,\ldots,w_r]} S'$$

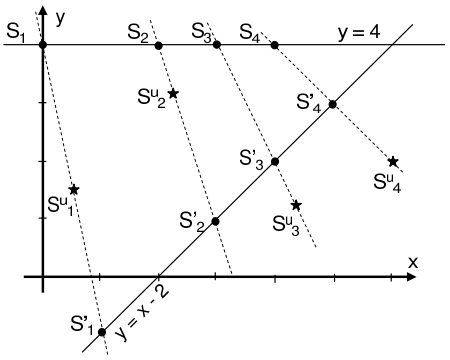

Figure 2: Example of union operation on two 4-point samples, S and S' with result S^u. The sample S satisfies the equation $y = 4$, and the sample S' satisfies the equation $y = x - 2$. The points in the resulting sample S^u are marked with stars and are chosen at random on the lines joining corresponding points in S and S'. All the points lie in the plane $z = 0$.

The union operation has a simple geometric intuition as shown in Figure 2 for 4-point samples. The sample S is obtained after the sequence of assignments "$y := 4; z := 0$" and S' after "$x := x + 1; y := x - 2; z := 0$". In both cases, the samples are obtained with the initial random values of x being 0, 2, 3 and 4. For each pair of corresponding points in the two samples, a point is chosen at random on the line joining the points. For example, the weight w_1 is 0.5 and thus S^u_1 is at midway between S_1 and S'_1. The weight factor can be greater than one or less than zero, as is the case for the third and fourth points. This example demonstrates two important aspects of the union operation. First, the states in both original samples satisfy the equation $z = 0$ (and no other common affine relationship among variables). Notice that S^u also satisfies this relationship. Thus, the union operation preserves common affine relationships. This is the completeness property that we state and prove below. The other notable aspect is that it is possible but highly unlikely for the S^u points to satisfy some other affine relationship (i.e., for all the points in S^u to be aligned). This is the probabilistic soundness property that we state precisely below.

An example using the union operation is shown in Figure 1 for a 1-point sample. The initial values of variables are not relevant because the variables are not live on input (we show these values in figure as $*$). The first union operation is performed with the weight $w_1 = 5$. Notice that the common relationship $a + b = 1$ is preserved after the first union operation. The second union operation is performed with the weight $w_2 = -3$. On the resulting state, we can verify the first assertion but not the second. In fact, we prove below that the probability that the second assertion would be satisfied accidentally is extremely small because there is at least one path on which it is not satisfied.

We conclude the discussion of the union operation with the statement and proof of the completeness and probabilistic soundness.

The completeness lemma for the union operation states that the resulting sample satisfies all affine relationships that

are satisfied by *both* of the original states.

LEMMA 1 (UNION COMPLETENESS LEMMA). *Let S and S' be two r-point samples whose points satisfy the affine equation $g = 0$. Then, for any choice of weights $w_1, \ldots, w_r$ the union $S^u = S \oplus_{[w_1,\ldots,w_r]} S'$ also satisfies the same equation.*

PROOF. It is easy to verify that if g is affine, then for any affine combination of two states $\rho \oplus_w \rho'$, we have $[\![g]\!](\rho \oplus_w \rho') = [\![g]\!]\rho \oplus_w [\![g]\!]\rho'$. Thus if the value of g is zero in both the states ρ and ρ', then the value of g is zero also on their affine combination. From here the completeness statement follows immediately. □

The following probabilistic soundness lemma for the union operation states that the probability of choosing the combination weights such that a new affine relationship is introduced is extremely small (for a large enough choice of weights).

LEMMA 2 (UNION SOUNDNESS LEMMA). *Let S and S' be two samples and let g be an expression such that $S \not\models g = 0$. More specifically, let t be the number of points in S that do not satisfy $g = 0$. Let $w_1, \ldots, w_r$ be chosen u.a.r. from $\tilde{F}$ and independently of each other and of the expression g. Let $S^u = S \oplus_{[w_1,\ldots,w_r]} S'$. Then the probability that $S^u \models g = 0$ is at most $(\frac{1}{d})^t$.*

PROOF. Without any loss of generality, let $S_1, \ldots, S_t$ be the t states in S that do not satisfy $g = 0$. For any $i \in 1 \ldots t$, consider the line joining the points S_i and S'_i. If $[\![g]\!]S_i = [\![g]\!]S'_i$, then this line is parallel to the hyperplane $g = 0$, and hence, no point on this line satisfies the equation $g = 0$. In other words, for any choice of w_i, $[\![g]\!]S^u_i = [\![g]\!]S_i = [\![g]\!]S'_i \neq 0$ and thus the probability that $S^u \models g = 0$ is zero. If on the other hand $[\![g]\!]S_i \neq [\![g]\!]S'_i$, then this line intersects the hyperplane $g = 0$ in exactly one point. In other words, there is only one choice for w_i (i.e., $\frac{[\![g]\!]S'_i}{[\![g]\!]S'_i - [\![g]\!]S_i}$) such that $[\![g]\!]S^u_i = 0$. Thus, the probability that the weight w_i was chosen such that state S^u_i satisfies the equation $g = 0$ is precisely $\frac{1}{d}$. Since $w_1, \ldots, w_t$ are all independent, it follows that the probability that all the states $S^u_1, \ldots, S^u_t$ satisfy the equation $g = 0$ is less than or equal to $(\frac{1}{d})^t$. This is an upper bound on the probability that all the states in S^u satisfy $g = 0$. □

5. THE INTERSECTION OPERATION

Next we consider the affine relationships that are introduced by conditionals. Consider the following program:

```
a := x + y ;
if x = y then b := a else b := 2 * x ;
assert (b = 2 * x)
```

The assert statement is true but in order to prove it we must notice that `x = y` in the true branch of the conditional. The random interpreter reaches the conditional with some sample. In order to reflect the conditional in the branches, we must change the sample (since all relationships are expressed implicitly as those satisfied by the sample). We could try to restart the interpretation with values that satisfy the conditional, but finding such initial values is hard. Or we could split the sample into two parts, one that satisfies the conditional and one that does not. But splitting is undesirable because working with smaller samples increases

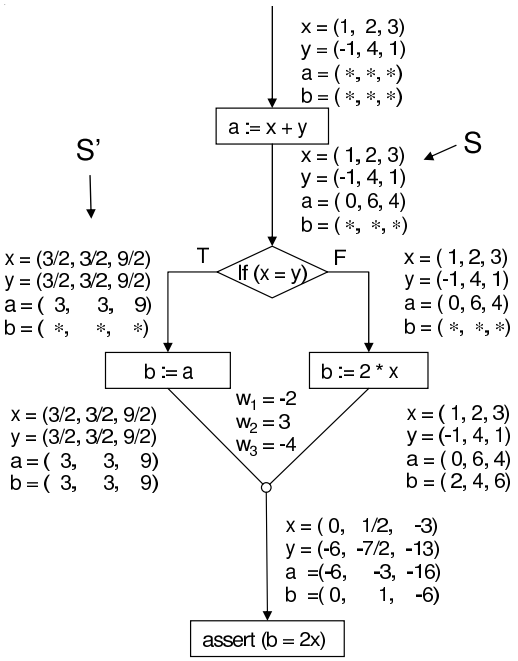

Figure 3: A code fragment showing the use of 3-point sample intersection. The numbers shown next to each edge represent random samples. Adjustment is used to obtain the sample S' from S, as detailed in Figure 4.

the probability that some accidental relationship holds. Furthermore, as we shall see, the probability that enough points satisfy the equality is extremely small anyway. Notice that we could not do something cheap such as replacing the occurrences of `x` with `y` in the true branch; this would not help in this case. We have to somehow adjust all of the previously computed variables as well, such as `a` in this example.

One of the novel aspects of this work is a procedure for transforming the sample in such a way that all of the previously existing relationships still hold, and additionally exactly one new relationship holds: the one given by the conditional. We do this by essentially "projecting" the sample points onto the plane given by the conditional. Orthogonal projection does not work since it destroys affine relationships. Instead we use the following function `Adjust`(S, e) to adjust the sample S according to the conditional $e = 0$:

```
Adjust(S, e) =
  Pick S_i and S_j in S such that [[e]]S_i ≠ [[e]]S_j .
  Pick w ∈ F such that ρ_0 = S_i ⊕_w S_j has the property
    that [[e]]ρ_0 ≠ 0 and [[e]]ρ_0 ≠ [[e]]S_k for all k ∈ {1...r}.
  For all k, let S'_k be the intersection of the plane
    e = 0 with the line passing through ρ_0 and S_k,
    i.e., S'_k = S_k ⊕_{w_k} ρ_0, where w_k = [[e]]ρ_0 / ([[e]]ρ_0 − [[e]]S_k) .
  The result is [S'_1, ..., S'_r].
```

An example of such an adjustment is shown in Figure 3. Here the program mentioned at the beginning of this section is executed on the 3-point sample shown at the top of the figure. Adjustment is used to obtain the sample S' from S. Notice that all of the states in S satisfy $a = x + y$ (due to the assignment). Now consider the distribution of the points in S when viewed inside the plane $a = x + y$ (as shown in

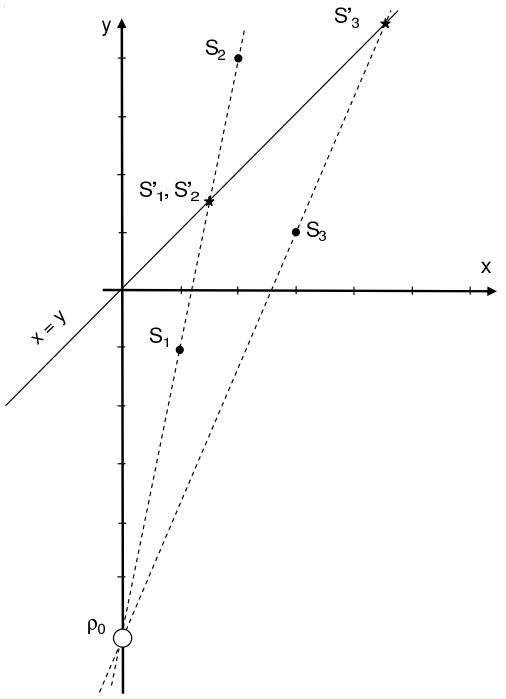

Figure 4: The detailed adjustment operation used in the example from Figure 3. The adjusted points are obtained as the intersections of the lines connecting the original points with ρ_0.

Figure 4). We pick the points S_1 and S_2 to play the role of S_i and S_j from the definition of `Adjust` (since the expression $x - y$ has different values on those points). Then we pick another point ρ_0 on the line determined by these two points. We picked ρ_0 at the intersection with the y axis but the only requirements for ρ_0 are that it is not in the plane $x - y = 0$, and that the lines passing through it and all the points in S are not parallel to the plane $x - y = 0$. Then we obtain the points $S'_k (k = 1, 2, 3)$ as the intersections of the lines that pass through ρ_0 and S_k with the plane $x = y$. Notice that two of the points will coincide.

The intuition behind this construction is that all of the points S'_k are obtained as affine combinations of three points S_i, S_j and S_k. As such they will satisfy all affine relationships between variables that are satisfied by all points in the original sample S. Furthermore, it is intuitive that since the original points are spread in the plane $a = x+y$ the resulting points are spread in the intersection of that plane with $x = y$ (with the exception that one of the points is "sacrificed" and will be equal to some other point).

Returning to the example from Figure 3, we see that the sample is adjusted only in the "true" branch of the equality conditional, since there is no affine equality that we can infer from a disequality. Notice that after adjustment the sample satisfies both the original relationships ($a = x + y$) and also the new one due to the conditional ($x = y$). Finally, the join operation is done using the random weights -2, 3 and -4 and the resulting sample now clearly reflects the desired assertion $b = 2 \times x$ (precisely because both sides of the join reflect the same assertion).

There are a few details in the definition of `Adjust` that deserve discussion. The first step of the algorithm presumes the existence of two points at which e has different values. If there are no such points, it means that e has the same value q on all the points in the sample. In such a case we need not perform any adjustment. Instead we declare that $e = q$ holds before the conditional and accordingly we consider only one branch depending on whether the constant q is zero or not. When this is the case we say that `Adjust` is not defined on the pair (S, e).

The second line in the `Adjust` algorithm finds a point on the line S_i to S_j that makes e have a non-zero value distinct from the values at the original points. Since e has different values at S_i and S_j this is always possible and finding such a value is a linear-time operation. Also, finding the intersection of a line with a plane is a simple computation.

To complete the discussion of the intersection operation, we state and prove below the completeness and then the soundness lemmas. The completeness lemma states that the adjusted sample satisfies all of the affine relationships satisfied by the original sample and satisfies also the relationship for which the original sample was adjusted.

LEMMA 3 (INTERSECTION COMPLETENESS LEMMA). *Let e and g be expressions and let S be a sample such that $S' = \texttt{Adjust}(S, e)$ is defined. Then for any choice of the intermediate point ρ_0 we have that $S' \models e = 0$ and if $S \models g = 0$ then $S' \models g = 0$.*

PROOF. By definition of S' we have that each S'_k from S' satisfies $e = 0$. Since ρ_0 is an affine combination of S_i and S_j (the two points picked in the first step of `Adjust`) then it satisfies all affine relationships that both S_i and S_j satisfy, hence also $g = 0$. Now each S'_k from S' is an affine combination of S_k and ρ_0 and therefore it also satisfies $g = 0$. □

The following soundness lemma implies that the adjusted sample satisfies exactly one more linearly independent affine relationship than the original sample.

LEMMA 4 (INTERSECTION SOUNDNESS LEMMA). *Let e and g be expressions and let S be a sample such that $\texttt{Adjust}(S, e)$ is defined. For any choice of the intermediate point ρ_0, there exists $\alpha \in \mathbb{F}$ such that if any t states in the sample $Adjust(S, e)$ satisfy the equation $g = 0$, then the corresponding t states in the sample S satisfy the equation $g + \alpha e = 0$.*

PROOF. Let $S' = Adjust(S, e)$ and let α be $-\frac{[\![g]\!]\rho_0}{[\![e]\!]\rho_0}$ (which is defined by the choice of ρ_0). Without any loss of generality, let $S'_1, \ldots, S'_t$ be the t states in the sample S' that satisfy the equation $g = 0$. For any $k \in \{1, \ldots, t\}$, we have that $S'_k = S_k \oplus_{w_k} \rho_0$, where $w_k = \frac{[\![e]\!]\rho_0}{[\![e]\!]\rho_0 - [\![e]\!]S_k}$. Since S'_k satisfies the equation $g = 0$, we can verify that S_k satisfies the equation $g - \frac{[\![g]\!]\rho_0}{[\![e]\!]\rho_0} e = 0$. This completes the proof. □

The geometric intuition behind the soundness lemma is that if some subset of the adjusted points lie in the hyperplane $g = 0$, then the corresponding subset of the original points lie in the hyperplane that contains the point ρ_0 and passes through the intersection of the hyperplanes $g = 0$ and $e = 0$. The soundness lemma implies that any equation $g = 0$ that is satisfied by the adjusted sample can be expressed as a linear combination of the equation $e = 0$ and some equation $g' = 0$ that is satisfied by the original sample.

Note that the soundness lemma indicates one such g' (i.e., $g + \alpha e$).

We can see from the proofs of the lemmas that as long as the adjusted points are affine combinations of the original points, the completeness lemma will be satisfied. Initially, we tried to compute affine combinations of just two points (by computing the intersection of the line they determine and $e = 0$). This made it impossible to state a clean soundness result in some corner cases; hence the three-point combination that we use at the moment. More complicated affine combinations could also be used and they might have the benefit of avoiding the overlap of two adjusted points.

Notice that during adjustment two distinct points in the original sample are transformed into the same point in the adjusted sample, thereby effectively reducing the number of points in a sample after each adjustment. This is not entirely unexpected. Since each adjustment "crowds" the sample into one fewer dimensions (e.g. from 3-d space into a 2-d plane), we expect that not all of the r points are going to remain independent. We shall see in Section 6 that because of this we need to start our random interpreter with a sample larger than the maximum number of adjustments on any path through the program. If we don't, then it is very likely that our algorithm will announce false relationships. For example, if we redo the example from Figure 3 with just the first two states (meaning that we ignore the third column in all samples), then after adjustment we will falsely infer many relationships, such as $a = 3$ and many others (there are many planes that pass through the point $S'_1 = S'_2$).

6. THE RANDOMIZED INTERPRETER

We now put together the ideas mentioned in the previous sections to define the randomized interpreter $\mathcal{R}$. For notational convenience, let us extend the definition of a sample as follows. A *sample* is either a sequence of r states or is undefined, in which case we write it as $\perp$. We also say that $\perp \models g = 0$ for any expression g. Essentially, $\mathcal{R}$ interprets a program like an abstract interpreter. The action of $\mathcal{R}$ over the various nodes of a flow-chart is defined below.

- Assignment Node: See Figure 5 (a).
  $S = \perp$, if $S' = \perp$.
  $S_i = S'_i[x \leftarrow [\![e]\!]S_i]$, otherwise.
- Conditional Node: See Figure 5 (b).
  $S^1 = \perp$ and $S^2 = \perp$, if $S' = \perp$
  $S^1 = S'$ and $S^2 = \perp$, if $S' \models e = 0$
  $S^1 = \perp$ and $S^2 = S'$, if $S' \models e - q = 0$ for some non-zero constant q
  $S^1 = \mathit{Adjust}(S', e)$ and $S^2 = S'$, otherwise
- Join Node: See Figure 5 (c).
  $S = S^1$, if $S^2 = \perp$
  $S = S^2$, if $S^1 = \perp$
  $S = S^1 \oplus_{[w_1,\ldots,w_r]} S^2$, otherwise, where $\{w_i\}$ are chosen independently and u.a.r. from $\tilde{F}$.

If $\mathcal{R}$ concludes that a conditional is always true (or always false), then it executes only the true (or false) branch of the conditional. Otherwise, it executes both branches of the conditional. To start the execution of the program, $\mathcal{R}$ chooses r points, each with n variables, independently and uniformly at random from $\tilde{F}^n$ as the initial sample.

The resulting samples can then be used to verify whether desired affine relationships hold at certain points in the program. Moreover, a sample can also be used to discover affine relationships by computing the affine subspace in which all the points of a sample lie. It is possible that the random interpreter is unsound, meaning that the resulting samples satisfy affine relationships that are false on some concrete execution. We prove in the rest of this section that the probability of this happening is extremely small, and can be reduced even further by repeating the experiment several times.

6.1 Error Probability Analysis

In our proof of an upper bound on the error probability we make use of the fact that $\mathbb{F}$ is a finite field. (However, we feel that it may be possible to prove similar results without this assumption). Working with a finite field is also desirable from an implementation point of view; otherwise the size of the values involved during the random interpretation doubles with each adjust operation. For these reasons we choose d to be a prime number and $\mathbb{F}$ to be the field of integers modulo d. We let $\tilde{F} = \mathbb{F}$ and thus d also represents the size of the field $\mathbb{F}$. The arithmetic operations over $\mathbb{F}$ are performed modulo the prime number d and division of a by b is implemented as multiplication of a by the multiplicative inverse of b in the field $\mathbb{F}$. However, our results are valid only if the concrete arithmetic operations of the program are interpreted over the same field $\mathbb{F}$ as opposed to the intended domain of integers. Thus we choose d larger that any value arising in a concrete execution of the program. For example, if the program operates on 32-bit numbers and if we make the assumption that its operations do not overflow, then we can choose d to be any prime larger than 2^{32}, and we make the interpreter use enough precision to be able to represent the entire field $\mathbb{F}$. In these conditions we can let $\tilde{F} = \mathbb{F}$.

For the purpose of the probability analysis we introduce an abstract interpreter $\mathcal{A}$ that computes a sound approximation of the set of affine relationships in a program. In the following definition we use the letter U to range over sets of affine relationships. We write $U \Rightarrow g = 0$ to say that the conjunction of the relationships in U imply $g = 0$. We write $U_1 \cap U_2$ for the set of relationships that are implied by both U_1 and U_2. (This operation is sometimes called the union of affine spaces [9]). Finally, we write $U[e/x]$ for the relationships that are obtained from those in U by substituting e for x. For notational convenience, we let $\perp$ represent an inconsistent set of affine relationships. We also say that $\perp \Rightarrow g = 0$ for any expression g. With these definitions we can define the action of $\mathcal{A}$ over the nodes of a flow-chart as follows:

- Assignment Node: See Figure 5 (a).
  $U = \perp$, if $U' = \perp$
  $U = \{x = e[x'/x]\} \cup U'[x'/x]$, otherwise, where x' is a fresh variable
- Conditional Node: See Figure 5 (b).
  $U^1 = U'$ and $U^2 = \perp$, if $U' \Rightarrow e = 0$
  $U^1 = \perp$ and $U^2 = U'$, if $U' \Rightarrow e - q = 0$ for some non-zero constant q
  $U^1 = U \cup \{e = 0\}$ and $U^2 = U'$, otherwise
- Join Node: See Figure 5 (c).
  $U = U^1$, if $U^2 = \perp$

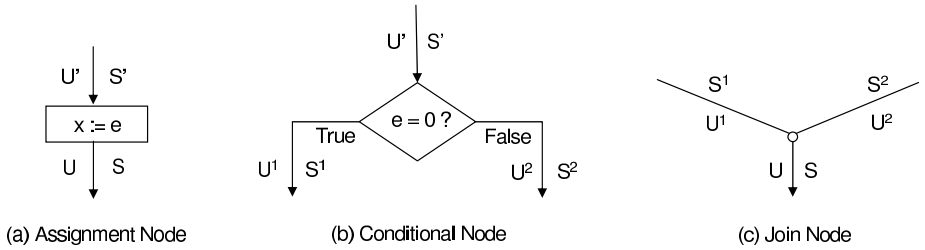

Figure 5: Flow-chart nodes

$U = U^2$, if $U^1 = \perp$
$U = U^1 \cap U^2$, otherwise

The abstract interpreter starts with the empty set of affine relationships between variables.

Implementations of abstract interpretations such as $\mathcal{A}$ have been described in the literature. The major concern there is the concrete representation of the set U and the implementation of the operation $U_1 \cap U_2$. For example, Karr's algorithm [9] uses the union of affine spaces, while Cousot's algorithm [5] uses convex hulls to implement the stronger operation that also handles affine inequalities. Here we use $\mathcal{A}$ only to state and prove the soundness and completeness results of the random interpreter $\mathcal{R}$.

Given a program, the sets of affine relationships U computed by $\mathcal{A}$ at every point in the program is uniquely defined. Corresponding to each such U, there is a random sample S (which depends on the random choices made by $\mathcal{R}$). We state below the relationship between U and S, in the form of completeness and soundness theorems.

THEOREM 5 (COMPLETENESS THEOREM). *Let U be a set of affine relationships computed by $\mathcal{A}$ at a given point in the program and let S be the corresponding sample. For any expression g, if $U \Rightarrow g = 0$, then $S \models g = 0$.*

The proof of Theorem 5 is based on Lemma 1 and Lemma 3, and is given in Appendix A.1. The completeness theorem implies that the random interpreter $\mathcal{R}$ discovers all the affine relationships that the abstract interpreter $\mathcal{A}$ discovers.

The next definition is necessary for the statement of the soundness theorem.

DEFINITION 6. *For any set of affine relationships $U \neq \perp$ computed by $\mathcal{A}$ for a program point, any subset $\tilde{S}$ of the random sample S computed by $\mathcal{R}$ for the same point, and any expression g such that $U \not\Rightarrow g = 0$, let $E(U, \tilde{S}, g)$ be the event that all points in $\tilde{S}$ satisfy the equation $g = 0$. Let $\Pr(E(U, \tilde{S}, g))$ denote the probability of this event over the random choices made by the random interpreter $\mathcal{R}$. Let $P(U, t) = max\{\Pr(E(U, \tilde{S}, g)) \mid \tilde{S} \subseteq S, |\tilde{S}| = t\}$.*

Note that the undesirable event $E(U, S, g)$ occurs only when the random interpreter claims that $g = 0$ holds but the abstract interpreter claims that it does not. The probability of this event is bounded by the quantity $P(U, r)$. The soundness theorem provides a bound for $P(U, r)$. However, in order for the inductive proof to work, the soundness theorem actually provides a bound for $P(U, t)$, where $1 \leq t \leq r$ is a parameter.

THEOREM 7 (SOUNDNESS THEOREM). *For any set of affine relationships U computed by A for a program point, $P(U, t) \leq (2d)^b \times (\frac{j+1}{d})^t$, where b and j are the maximum number of intersection (branches) and union (join) operations respectively performed by $\mathcal{A}$ on any path before computing U.*

According to Theorem 7, given any relationship not verified by the abstract interpreter, the probability (over the random choices made by the random interpreter) that the relationship is verified by the random interpreter is extremely small. The proof of Theorem 7 is non-trivial, but is easy to follow once the reader is comfortable with the above notation. The proof is given in Appendix A.2. We use this soundness theorem in the next section to prove Theorem 8, which establishes an upper bound on the probability that $\mathcal{R}$ is unsound even for programs involving loops.

The bound on $P(U, t)$ can be interpreted as follows. Each intersection operation increases the probability of error by a factor of $2d$ while all the j join operations together contribute a factor of $(\frac{j+1}{d})^t$. The latter can be explained in a rather interesting manner. If $\mathcal{R}$ does not perform any adjust operations ($b = 0$), then it can be viewed as evaluating multivariate polynomials of degree $j+1$ whose variables are the input variables of the program along with a variable corresponding to each join operation. For example, in Figure 1 the multivariate polynomial corresponding to variable a is $w_1(0) + (1 - w_1)(1) \equiv 1 - w_1$, and that corresponding to b is $w_1(1) + (1 - w_1)(0) \equiv w_1$. Similarly the polynomial corresponding to variable c is $w_2(b - a) + (1 - w_2)(2a + b) \equiv w_2(w_1 - 1 + w_1) + (1 - w_2)(2 - 2w_1 + w_1) \equiv 3w_1w_2 - w_1 - 3w_2 + 2$ and that corresponding to d is $w_2(1 - 2b) + (1 - w_2)(b - 2) \equiv w_2(1 - 2w_1) + (1 - w_2)(w_1 - 2) \equiv -3w_1w_2 + w_1 + 3w_2 - 2$. Note that the multivariate polynomial corresponding to the expression $c + d$ is identically equal to 0 and hence $\mathcal{R}$ always succeeds in verifying the assertion $c + d = 0$, while the multivariate polynomial corresponding to the expression $c - a - 1$ is $3w_1w_2 - 3w_2$ which is not identically equal to 0 and hence $\mathcal{R}$ declares the assertion $c - a - 1 = 0$ to be invalid with high probability. The classic random testing procedure to check whether a multivariate polynomial of degree $j + 1$ is identically equal to 0 or not has an error probability at most $(\frac{j+1}{d})^t$, where d is the size of the set from which random values are chosen and t is the number of trials [13]. This is precisely the factor contributed by j join operations to the bound for $P(U, t)$.

For the informal intuition behind the factor corresponding to intersections consider the case when $j = 0$. Then each adjust operation has the effect of reducing the number of useful points by 1 (along with an additional factor of 2 that is necessary to accommodate some corner cases).

6.2 Fixed Point Computation

The lattice of sets of affine relationships (under the set union operation, and the intersection operation as described in Section 6.1) has finite depth n since there can be at most n linearly independent affine relationships involving n variables. Thus the abstract interpreter $\mathcal{A}$ interpreting a program with loops is guaranteed to reach a fixed point. Given the close relationship between $\mathcal{A}$ and $\mathcal{R}$ as mentioned in Theorem 5 and Theorem 7, $\mathcal{R}$ also reaches a fixed point with high probability.

One way to detect when $\mathcal{R}$ has reached the fixed point is to compare the rank of the samples (viewed as matrices) at relevant locations in two successive iterations of a loop. The rank of a sample is always equal to the number of variables minus the number of linearly independent relationships that the sample satisfies. Thus, if the rank has stabilized, the number of linearly independent relationships satisfied by S has been stabilized, and so has the set of affine relationships satisfied by S.

We now state and prove the final result that bounds the probability of error in the operation of $\mathcal{R}$. In particular, it also bounds the error probability that $\mathcal{R}$ does not reach a fixed point, or that the rank testing mechanism fails to correctly detect the fixed point.

THEOREM 8 (PROBABILISTIC SOUNDNESS THEOREM). *The probability that some random sample S models a relationship $g = 0$ that is not implied by its corresponding set of affine relationships U, is at most $m \times 2d^n \times (2d)^b \times (\frac{j+1}{d})^r$, where m is the total number of operations performed by $\mathcal{A}$ before reaching the fixed point (among which there are b intersections and j union operations).*

PROOF. Consider one particular sample S and the corresponding set of affine relationships U. There are less than $2d^n$ different affine relationships with coefficients from $\mathbb{F}$ between the n variables and hence this is an upper bound on the number of relationships not implied by U. Thus, it follows from Theorem 7 that the probability that there exists an expression g such that $S \models g = 0$ and $U \not\models g = 0$ is at most $2d^n \times (2d)^b \times (\frac{j+1}{d})^r$. And since there are m such samples, the desired result follows easily. □

Note that if all samples S model only relationships that are implied by the corresponding set of affine relationships U, then $\mathcal{R}$ reaches a fixed point when $\mathcal{A}$ does so, and the rank testing mechanism faithfully tells whether or not the fixed point has been reached. Theorem 8 says that the probability that this happens is high.

Theorem 8 implies that r must be $O(n + b)$ to achieve a small error probability. In particular, if $d > max\{j^3, 2m, 8\}$ and we pick $r > 1.5(n + 1) + 2b$, then the error probability is bounded above by $(\frac{1}{d})^{\frac{2}{3}(r-1.5(n+1)-2b)}$.

6.3 Computational Complexity

The cost of each intersection and union operation performed by the randomized interpreter is $O(nr)$. Each assignment operation takes $O(r)$ time assuming that each assignment operation involves a constant number of arithmetic operations. On the other hand, Karr's deterministic algorithm [9] incurs a cost of $O(n^3)$ for each union operation and $O(n^2)$ for each intersection operations.

Computing the rank of a sample takes $O(n^3)$ time. This may be expensive considering that the other operations performed by $\mathcal{R}$ are at most $O(nr)$. An alternative way to ensure that a fixed point was reached is to estimate an upper bound on the number of iterations required by $\mathcal{A}$ to reach a fixed point for a loop and then let $\mathcal{R}$ go around that loop that many number of times. Note that since n is the depth of the lattice of the sets of affine relationships, it determines an upper bound on the number of iterations required to reach a fixed point.

7. BEYOND AFFINE EQUALITIES

The language of expressions that we have considered so far allows only for affine arithmetic expressions. In this section we speculate about the uses of randomized algorithms for programs containing other features as well.

We have experimented successfully with the algorithm presented here even for arithmetic expressions that are non-linear. We believe that the probabilistic soundness results still hold but we have yet to prove this formally. Another difficulty is that completeness would be seriously compromised by the union operation and it is not clear how to implement efficiently an appropriate intersection operation. In the absence of union and intersections (e.g. for a basic-block analysis) both completeness and probabilistic soundness can be attained. Consider, for example, the following basic block:

```
a := (y + 1) × (y - 1) ;
b := y × y - 1;
assert (a = b)
```

To prove the assert statement, we need to prove that $(y + 1) \times (y - 1) \equiv y \times y - 1$. There is no known deterministic polynomial-time algorithm to solve the above problem (the full monomial expansion of a polynomial can be exponential in the size of the original polynomial). However, there is a very simple and fast randomized algorithm (see [13]) that is complete and probabilistically sound: compare the values of the two polynomial expressions on a few random inputs.

We consider in this paper only equality and disequality conditionals. The algorithm can be applied to other kinds of conditionals, by not doing any adjustment. The soundness results still hold but this weakens the analysis since it is not able to exploit the information from such conditionals. In future work we plan to explore alternatives for the union and intersection procedures that work with inequalities as well. Interestingly, if we restrict the weights to the range $[0, 1]$ then not only affine equations but also affine inequalities are preserved.

It is worth mentioning that the random interpreter approach seems to be ill suited for verifying disequalities. For example, just because a number of random runs for a program yield non-zero results we cannot conclude that there is no run in which the result is zero. However, we can infer some disequalities that are consequences of affine equalities. For example, if x, y and z are integer variables and $x-y = a\times z+b$ such that either $a = 0$ and $b \neq 0$ or $0 < b < a$ (for some integer constants a and b), then we know that it cannot be the case that x and y are equal.

The analysis that we have described is a path-insensitive analysis. However, by choosing the random values for each join carefully, we can capture some path-sensitive information. For example, consider verifying the assert statement at the end of the following program:

```
if (x = y) then a := 1 else a := 4;
if (x = y) then b := 2 else b := 5;
assert (b = a + 1)
```

If $\mathcal{R}$ chooses different random values for the two joins in the above program, it fails to verify the assert statement. However, if the same random values are chosen for the two joins, $\mathcal{R}$ is able to verify the assert statement. Note that choosing same random values for joins that correspond to equivalent conditionals does not break any of our probabilistic soundness results.

8. COMPARISON WITH RELATED WORK

Blum, Chandra and Wegman [3] showed how to compute fingerprints of read-once branching programs in order to decide their equivalence in probabilistically polynomial time. Their idea was to assign random values to boolean variables instead of the usual 0 or 1 boolean values and then *evaluate* the branching program by performing multiplication in place of conjunction, addition in place of disjunction and subtraction from 1 in place of negation. Our technique for handling joins is reminiscent of their idea, where we also assign a random value instead of the typical 0 or 1 boolean values at join points.

Aiken, Fähndrich and Su [1] have used random sampling for race detection in Relay Ladder Logic programs with probabilistic guarantees. Fähndrich, Foster, Su and Aiken [7] have used randomization for efficiently solving general inclusion constraints in the context of pointer analysis for C programs.

Value numbering is a technique whereby hash values are assigned to expressions and variables with equivalent hash values are declared to be equal [12]. However, the problem with this technique is that it is very closely tied to the structure of expressions rather than their semantics. For example it cannot detect that $(x + y) + z = (z + x) + y$. In some sense, a sample can be thought of as a set of hash values for the program variables at that location, except that we can maintain it across assignments, union and intersection operations. Wegman, Sreedhar and Bodik [14] have independently extended value numbering to be less sensitive to the syntax of the expressions. They are also using an affine combination for joins but do not have an intersection operation.

Random testing [8] is most commonly used to verify assertions in a program, and this technique has recently been used to discover useful invariants from program traces [6]. However, the greatest problem with this kind of approach is its lack of soundness since it is only practical to explore a limited number of paths. Our technique is similar in spirit to this technique but avoids these problems by executing both sides of a branch (and locally adjusting the values of variables to account for the latest path predicate) and then merging the results at join points.

Symbolic analysis techniques have also been used to discover linear relationships among variables. For example, Karr [9] describes an abstract interpretation on a lattice of affine relationships between variables. His analysis is able to infer the same relationships as the one presented in this paper, in the case when the program has only linear computation. In the presence of non-linear computations our algorithm is slightly more complete as explained in Section 7. Karr's algorithm works on a lattice of finite depth whose union and intersection operations require $O(n^3)$ and $O(n^2)$ arithmetic operations respectively while our algorithm requires $O(nr)$ arithmetic operations for both joins and intersection. The real complexity however in Karr's algorithm is in the implementation. The computation of an affine union of spaces is significantly more involved than our join operation. Just like for our algorithm, their union and intersection operations require multiplication of two numbers of the same size. Although the paper is silent about this aspect, an implementation of the algorithm must deal with exponentially large numbers. The abstract interpretation algorithm used by Cousot and Halbwachs [5] goes a step further and discovers also linear inequality relationships among variables. This algorithm also appears to suffer from the presence of exponentially large numbers.

It is interesting to compare the random interpreter approach described in this paper with abstract interpretation [4] in general. In both cases there are union and intersection operations (called join and meet in abstract interpretation). The samples used by the random interpreter are representatives of the sets of affine relationships (satisfied by them) which form a lattice (under the set union operation, and the intersection operation as described in Section 6.1). However, on rare occasions, the random interpreter might perform an unsound join operation (returning a sample whose representative is not greater than the lowest upper bound of the representatives of the joined samples).

9. CONCLUSION

We have presented in this paper the preliminary results of our investigation of the use of randomized algorithms in program analysis. We have found that by running the program on random inputs and by relaxing the semantics of the conditionals, such that we execute both branches of each conditional, we can quickly compute a "fingerprint" of the program that reflects affine invariants of the program and, with high probability, nothing more. The surprising result is that with this form of testing, a single run through the program captures information about all the possible paths, thus making it possible to filter out quickly the invariants that hold only on certain paths.

The algorithms discussed in this paper are a selection from a set of algorithms that we are exploring. A general characteristic of the progress in our project has been that the algorithms for random interpretation are fairly easy to design and most often trivial to implement. In each case our intuition suggests that the probability of unsound results is extremely small, and in fact experiments do not reveal any unsoundness. But proving an upper bound for the probability of unsoundness has been an extremely challenging task, and most often we have to settle with conservative bounds. Nevertheless, we believe that randomization has much to offer to program analysis and this area is worth of future research.

Program analysis is provably hard, and we have all learned not to expect perfect results. However, this attitude has manifested itself mostly in a large number of static analysis approaches in which completeness is sacrificed and false positives are accepted as a fact of life, while soundness remains the *sine qua non* of program analysis. The results of this paper show that it might be profitable to relax this strict view of soundness, and trade off minuscule amounts of soundness in return for other advantages such as better computational complexity, or simplicity, or even more precise results. And when we think that in the grand scheme of things, program analyses are used to produce software that interacts with other potentially buggy libraries running on

fallible hardware, we realize that maybe a minuscule probability of unsoundness in the analysis is tolerable after all.

10. ACKNOWLEDGMENTS

We would like to thank Mark Wegman for providing useful directions and helpful discussions. We would also like to thank the members of the Open Source Quality Group for carefully reviewing early drafts and for helping us improve the paper.

11. REFERENCES

[1] A. Aiken, M. Fähndrich, and Z. Su. Detecting races in relay ladder logic programs. *International Journal on Software Tools for Technology Transfer*, 3(1):93–105, 2000.

[2] B. Alpern, M. Wegman, and F. Zadeck. Detecting equality of variables in programs. In *Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 1–11, Jan. 1988.

[3] M. Blum, A. Chandra, and M. Wegman. Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, 10:80–82, 1980.

[4] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*, pages 234–252, 1977.

[5] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM Symposium on Principles of Programming Languages*, pages 84–97, 1978.

[6] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):1–25, Feb. 2001.

[7] M. Fähndrich, J. Foster, Z. Su, and A. Aiken. Partial online cycle elimination in inclusion constraint graphs. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 85–96, Montreal, CA, June 1998.

[8] D. Hamlet. Random testing. In J. Marciniak, editor, *Encyclopedia of Software Engineering*, pages 970–978. Wiley, New York, 1994.

[9] M. Karr. Affine relationships among variables of a program. In *Acta Informatica*, pages 133–151. Springer, 1976.

[10] G. C. Necula. Translation validation for an optimizing compiler. In *ACM SIGPLAN '00 Conference on Programming Language Design and Implementation (PDLI)*, pages 83–94, Vancouver, BC, Canada, 18–21 June 2000. ACM SIGPLAN.

[11] A. Pnueli, M. Siegel, and E. Singerman. Translation validation. In B. Steffen, editor, *Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS '98*, volume LNCS 1384, pages 151–166. Springer, 1998.

[12] B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Global value numbers and redundant computations. In *POPL '88. Proceedings of the conference on Principles of programming languages*, pages 12–27, Jan. 1988.

[13] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *JACM*, 27(4):701–717, Oct. 1980.

[14] M. Wegman, V. C. Sreedhar, and R. Bodik. Probabilistic value numbering. Unpublished manuscript, 2001.

APPENDIX

A. PROOF OF SOUNDNESS AND COMPLETENESS THEOREMS

We now give the proofs for the completeness and soundness theorems stated in Section 6. Both the abstract interpreter $\mathcal{A}$ and the randomized interpreter $\mathcal{R}$ perform similar operations for each node in the flow-graph. The proofs are by induction on the number of operations performed by the interpreters. The computation performed by the interpreters can be viewed as going forward in the sense that the outputs of a flowchart node are determined by the inputs of the node. Hence, for the inductive case of the proof, we prove that the required property holds for the outputs of the node given that it holds for the inputs of the node.

A.1 Proof of Completeness

The proof is by induction on the number of operations performed by the interpreters. The base case is trivial since initially $U = \emptyset$ and hence for any expression g, $U \not\Rightarrow g = 0$. For the inductive case, the following scenarios arise. Let g be any expression.

- Assignment Node: See Figure 5 (a).
  Assume that $U \Rightarrow g = 0$. We prove that $S \models g = 0$. Consider the expression $g' = g[e/x]$. Since $U \Rightarrow g = 0$, $U' \Rightarrow g' = 0$. It follows from the induction hypothesis on U' and S' that $S' \models g' = 0$. Hence, $S \models g = 0$.
- Conditional Node. See Figure 5 (b).
  We prove that
  (a) if $U^1 \Rightarrow g = 0$, then $S^1 \models g = 0$, and
  (b) if $U^2 \Rightarrow g = 0$, then $S^2 \models g = 0$.
  Three possibilities arise here.

  (i) $U' \Rightarrow e = 0$. It follows from induction hypothesis on U' and S' that $S' \models e = 0$. By definition of $\mathcal{A}$, $U^1 = U'$ and $U^2 = \bot$. Similarly, $S^1 = S'$ and $S^2 = \bot$.
  (a) Assume that $U^1 \Rightarrow g = 0$. Thus, $U' \Rightarrow g = 0$ and by induction hypothesis $S' \models g = 0$. Thus, $S^1 \models g = 0$.
  (b) The proof obligation for S^2 follows immediately.

  (ii) $U' \Rightarrow e - q = 0$ for some non-zero constant q. It follows from induction hypothesis on U' and S' that $S' \models e - q = 0$. By definition, $U^1 = \bot$ and $U^2 = U'$. Similarly, $S^1 = \bot$ and $S^2 = S'$. The proof for this case is similar to the symmetric case shown above.

  (iii) $U' \not\Rightarrow e - q = 0$ for any constant q.
  By definition, $U^1 = U' \cup \{e = 0\}$ and $U^2 = U'$.
  (a) Assume that $U^1 \Rightarrow g = 0$. There exists an expression g' such that $U' \Rightarrow g' = 0$ and $g = g' + \lambda e$ for some constant λ. It follows from induction hypothesis on U' and S' that $S' \models g' = 0$. It follows from Lemma 3 that $S^1 \models g' = 0$ and $S^1 \models e = 0$. Hence, $S^1 \models g' + \lambda e = 0$.
  (b) Assume that $U^2 \Rightarrow g = 0$. Thus, $U' \Rightarrow g = 0$. It follows from induction hypothesis on U' and S' that $S' \models g = 0$. Either $S^2 = S'$ or $S^2 = \bot$. In either case, $S^2 \models g = 0$.

- Join Node: See Figure 5 (c).
  Assume that $U \Rightarrow g = 0$. We prove that $S \models g = 0$. By definition of $\mathcal{A}$, $U^1 \Rightarrow g = 0$ and $U^2 \Rightarrow g = 0$. By induction hypothesis on U^1 and S^1 and on U^2 and S^2, we have that $S^1 \models g = 0$ and $S^2 \models g = 0$. It now follows from Lemma 1 that $S \models g = 0$.

A.2 Proof of Soundness

The proof is again by induction on the number of operations performed by the interpreters. For the base case, $j = b = 0$ and $U = \emptyset$. Let g be any expression which is identically not equal to 0. We have $S = S^0$, all of whose states are chosen independently and u.a.r. from $\tilde{F}^n$. The probability that a particular point in sample S^0 satisfies the equation $g = 0$ is at most $\frac{1}{d}$. Thus, the probability that some particular t points from sample S^0 satisfy the equation $g = 0$ is at most $(\frac{1}{d})^t$. Hence, $P(U, t) \leq (\frac{1}{d})^t$.

For the inductive case, the following scenarios arise.

- Assignment Node: See Figure 5 (a).
  We prove that $P(U, t) = P(U', t)$ and the result follows from induction hypothesis on U' and S'.
  Let g be any expression such that $U \not\Rightarrow g = 0$. Consider the expression $g' = g[e/x]$. Let $\tilde{S}$ be any subset of sample S. Let $\tilde{S}'$ be the corresponding subset of sample S'. Note that event $E(U, \tilde{S}, g)$ happens iff $E(U', \tilde{S}', g')$ happens. Thus,
  $P(U, t) = P(U', t)$.

- Conditional Node: See Figure 5 (b).
  We prove that
  (a) $P(U^1, t) \leq (2d) \times P(U', t)$, and
  (b) $P(U^2, t) \leq (2d) \times P(U', t)$
  and the result follows from the induction hypothesis on U^1 and S^1, and on U^2 and S^2. The following possibilities arise:

  (i) $U' \Rightarrow e = 0$
  It follows from Theorem 5 that $S' \models e = 0$.
  (a) $U^1 = U'$. $S^1 = S'$. Thus, $P(U^1, t) = P(U', t)$.
  (b) $U^2 = \perp$. Thus, $P(U^2, t) = 0$.

  (ii) $U' \Rightarrow e - q = 0$ for some non-zero constant q
  It follows from Theorem 5 that $S' \models e - q = 0$. Hence,
  (a) $U^1 = \perp$. Thus, $P(U^1, t) = 0$
  (b) $U^2 = U'$ and $S^2 = S'$. Thus, $P(U^2, t) = P(U', t)$.

  (iii) $U' \not\Rightarrow e - q = 0$ for any constant q
  It is possible that $S' \models e - q = 0$ for some constant q (note that Theorem 5 does not help us here). This may be a cause of unsoundness and hence we consider this possibility while computing $P(U^1, t)$ and $P(U^2, t)$ below.
  (a) $U^1 = U' \cup \{e = 0\}$. Let g be any expression such that $U^1 \not\Rightarrow g = 0$. Note that $U' \not\Rightarrow g + \lambda e = 0$ for any constant λ. Let $\tilde{S}^1$ be any subset of t points of sample S^1. Let $\tilde{S}'$ be the corresponding subset of sample S'. The event $E(U^1, \tilde{S}^1, g)$ occurs only if $S' \models e - q = 0$ for some constant q, or $S' \not\models e - q = 0$ for any constant q (hence, Adjust(S',e) is defined) and the event $E(U', \tilde{S}', g + \lambda e)$ occurs for any constant λ (this follows from Lemma 4). Thus, $\Pr(E(U^1, \tilde{S}^1, g))$
  $\leq \sum_{q \in \mathbb{F}} \Pr(S' \models e - q = 0) + \sum_{\lambda \in \mathbb{F}} \Pr(E(U', \tilde{S}', g + \lambda e))$
  $\leq \sum_{q=0}^{d-1} P(U', r) + \sum_{\lambda=0}^{d-1} P(U', t)$
  $= d \times P(U', r) + d \times P(U', t)$
  $\leq (2d) \times P(U', t)$
  Hence, $P(U^1, t) \leq (2d) \times P(U', t)$
  (b) $U^2 = U'$. Let g be any expression such that $U^2 \not\Rightarrow g = 0$. Let $\tilde{S}^2$ be any subset of t points of sample S^2. Let $\tilde{S}'$ be the corresponding subset of sample S'. The event $E(U^2, \tilde{S}^2, g)$ occurs only if $S' \models e = 0$, or $S' \not\models e = 0$ (hence $S^2 = S'$) and the event $E(U', \tilde{S}', g)$ occurs. Thus,

  $$\begin{aligned}\Pr(E(U^2, \tilde{S}^2, g)) &\leq \Pr(S' \models e = 0) + \Pr(E(U', \tilde{S}', g)) \\ &\leq P(U', r) + P(U', t) \\ &\leq 2P(U', t) \ \leq \ (2d) \times P(U', t)\end{aligned}$$

  Hence, $P(U^2, t) \leq (2d) \times P(U', t)$

- Join Node: See Figure 5 (c).
  We prove that $P(U, t) \leq (2d)^t \times (\frac{j+1}{d})^t$, where b and j are the maximum number of intersection and union operations performed by $\mathcal{A}$ for computing U. Let b_1 and j_1 be the maximum number of intersection and union operations performed by $\mathcal{A}$ for computing U^1. Let b_2 and j_2 be the maximum number of intersection and union operations performed by $\mathcal{A}$ for computing U^2. Clearly, $j = 1 + max(j_1, j_2)$ and $b = max(b_1, b_2)$. Let g be any expression such that $U \not\Rightarrow g = 0$. Thus, either $U^1 \not\Rightarrow g = 0$ or $U^2 \not\Rightarrow g = 0$. Consider the case when $U^1 \not\Rightarrow g = 0$. (The other case is symmetric). Let $\tilde{S}$ be any subset of t points of sample S. We are going to compute the probability that $\tilde{S} \models g = 0$ (or, $\Pr(E(U, \tilde{S}, g))$). Let $\tilde{S}^1$ be the corresponding subset of t points of sample S^1. For $i \in \{0, \ldots, t\}$, let event E_i be the event that *exactly* i states in $\tilde{S}^1$ satisfy the equation $g = 0$. The events E_i form a disjoint partition of the event space. This means that:

  $$\Pr(E(U, \tilde{S}, g)) = \sum_{i=0}^{t} \Pr(E_i) \times \Pr(E(U, \tilde{S}, g) \mid E_i)$$

  By Lemma 2, $\Pr(E(U, \tilde{S}, g) \mid E_i)$ (the probability that $\tilde{S} \models g = 0$ if it is known that exactly i states in $\tilde{S}^1$ satisfy $g = 0$, or equivalently that $t - i$ states do not satisfy $g = 0$) is at most $(\frac{1}{d})^{t-i}$.
  In order to compute $\Pr(E_i)$ we observe that there are $\binom{t}{i}$ ways to choose a subset of i states from $\tilde{S}^1$. For each such subset, the probability that it satisfies $g = 0$ is at most $(2d)^{b_1} \times (\frac{j_1+1}{d})^i$ (by induction hypothesis). In conclusion, $\Pr(E_i) \leq \binom{t}{i} \times (2d)^{b_1} \times (\frac{j_1+1}{d})^i$.

  $$\begin{aligned}\text{Thus, } \Pr(E(U, \tilde{S}, g)) &= \sum_{i=0}^{t} \Pr(E_i) \times \Pr(E(U, \tilde{S}, g) \mid E_i) \\ &\leq \sum_{i=0}^{t} \binom{t}{i} (2d)^{b_1} (\tfrac{j_1+1}{d})^i \times (\tfrac{1}{d})^{t-i} \\ &= (2d)^{b_1} \times (\tfrac{j_1+2}{d})^t \\ &\leq (2d)^b \times (\tfrac{j+1}{d})^t\end{aligned}$$

  Thus, $P(U, t) \leq (2d)^b \times (\frac{j+1}{d})^t$.