In Defense of Probabilistic Static Analysis

Benjamin Livshits
Microsoft Research
livshits@microsoft.com

Shuvendu K. Lahiri
Microsoft Research
shuvendu@microsoft.com

1. Introduction

In designing a static analysis, one has to trade-off analysis scalability for analysis precision. While much has been said about clever and useful ways to create static approximations of dynamic program behavior, at the end of the day, static analysis precision is not elastic. In other words, the analysis designer either over-provisions, designing an algorithm that overly precise, suffering in terms of scalability, or what is often even worse, under-provisions, designing an algorithm that is insufficiently precise, often yielding an analysis that is too inaccurate to be deployed.

Moreover, much of the time, precision of a given static analysis is both over- and under-provisioned. This is because precision is highly dependent on the language features and how (frequently) they are used in the programs of interest. For instance, 1-level object sensitivity is overkill for most types of code, except it is needed for programs that use factories extensively. Moreover, often, 2-level object sensitivity would be required to handle object factories more accurately.

The lack of elasticity in terms of static analysis precision has lead analysis designers to combine or *stage* their analysis together. For example, reflection analysis and string analysis are frequently combined with a points-to and analysis of JNI constructs. For C, a pre-processing step may be used to identify various allocation functions such as `malloc`, `kalloc`, etc. In JavaScript, aliases to `eval` may need to be identified ahead of other analysis steps. Widely usable analyses are therefore constructed piecemeal, ultimately resulting in a patchwork of different analysis techniques connected together.

2. Elasticity

In this paper we claim that we need a more elastic approach to analysis precision. The way to achieve elasticity that we advocate is probabilistic static analyses. As a simple illustrative example, consider a static pointer analysis that computes relation *pointsTo(v, h)*, to tell whether variable *v* may point to heap object *h*. A probabilistic version of this analysis will also include p as in *pointsTo(p, v, h)*; p is the probability associated with this points-to fact. Note that this change of perspective shifts our analysis away from soundness; i.e. if the original points-to analysis was in fact sound, the resulting one will not generally not yield precise facts, instead making it possible to make probabilistic judgments.

3. Benefits

When it comes to interpreting analysis results, we often observe fuzzy treatment: not all results are treated the same. In fact, giving priorities or warning levels to analysis results is an ad-hoc form of introducing probabilistic treatment, designed to recover some of the imprecision when it comes to end-user interactions. We, however, believe that intrinsically building probabilistic reasoning into static analysis design leads to a number of desirable outcomes.

- Static analysis results can be naturally ranked or prioritized in terms of certainty, nearly a requirement in a situation where analysis users are frequently flooded with results (result prioritization).
- Program points or even static analysis inference rules and facts leading to imprecision can be identified with the help of backward propagation (blame assignment and analysis debugging).
- In an effort to make their analysis fully sound, analysis designers often combine certain inference rules with those that cover generally unlikely cases to maintain soundness. For instance, an analysis may have a hard rule about value flow and a soft rule that corresponds to the relatively unlikely flow of data via exceptions. Naturally blending such inference rules together, by giving high probabilities to the former and low probabilities to the latter allows us to balance soundness and utility considerations.
- End-quality of analysis results can often be improved by domain knowledge such as information about variable naming, check-in information from source control repositories, historical data from bug repositories, etc. Often times, this kind of data is used to prioritize or suppress analysis results. A better approach involves incorporating this information directly, in the form of priors, making it explicit in the tool.

In short, probabilistic static analysis gives a fresh new way to look at very old problems and naturally provides solutions to problems that have been difficult to address in the past.

4. Motivating example

We show the power of probabilistic thinking using a simple NULL detection analysis. The goal is to find NULL dereference errors. To simplify maters, we will address direct flow, object allocation and NULL, omitting some of the less relevant language constructs. Reasoning about such code can be expressed in Datalog using a handful of inference rules. Rules on lines 1 and 2 show two cases of flow propagation: the traditional transitive case on line 1 and a special form of conditional assignments on line 2. Conditional assignment relation ASSIGNCOND models assignments that are guarded by a conditional at runtime.

```
// transitive flow propagation
1.   FLOW(x,z):- FLOW(x,y), ASSIGN(y,z)
2.   FLOW(a,c) :- FLOW(a,b), ASSIGNCOND(b,c)
3.   FLOW(x,x).

// nullable variables
4.   NULLABLE(x) :- FLOW(x,y), ISNULL(y)

// error detection
5.   ERROR(a) :- ISNULL(a), DEREF(a)
6.   ERROR(a) :- !ISNULL(a), NULLABLE(a), DEREF(a)
```

Relation NULLABLE corresponds to variables that can assume NULL as a possible value, whereas ISNULL is a given fact about variables that are initialized to NULL in the program. Finally, rules on lines 5 and 6 identify possible errors that stem from dereferencing NULL values.

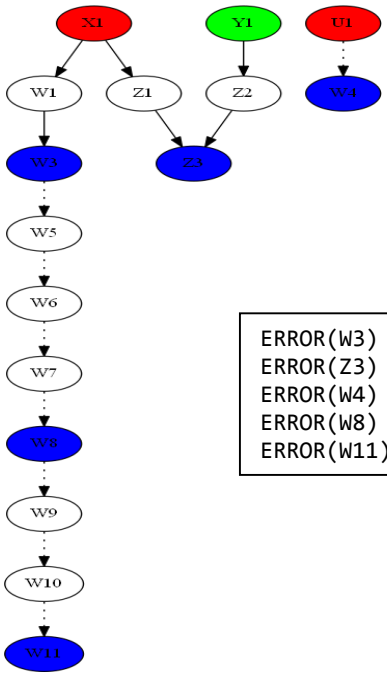
Note that rules on lines 1 and 2 are different in that 2 is significantly less likely than one, which is a *certain* rule. Similarly, 4 is a *certain* rule and 5 is not necessarily as likely. Our insight is that modeling these levels of certainly is helpful for analysis quality.
To add probabilities to these rules in this paper, we chose to encode static analysis problems in the form of Markov logic using Alchemy, a Markov logic tool from the University of Washington.

We show Alchemy rules below. Note that for the most part they directly correspond to Datalog rules, except the implication notation goes left-to-right (antecedent => consequent as opposed consequent :- antecedent). Additionally, Alchemy support more of first-order logic than most Datalog systems.

```
// transitive flow propagation
1.   FLOW(x,y) ^ ASSIGN(y,z) => FLOW(x,z).
2.   1 FLOW(a,b) ^ ASSIGNCOND(b,c) => FLOW(a,c)
3.   FLOW(x,x).

// nullable variables
4.   FLOW(x,y) ^ ISNULL(y) => NULLABLE(x).

// error detection
5.   ISNULL(a)^ DEREF(a) => ERROR(a).
6.   0.5 !ISNULL(a) ^ NULLABLE(a) ^ DEREF(a)
     => ERROR(a)

// priors and shaping distributions
7.   3 !FLOW(x,y)
```

We add weights to rules on lines 2 and 6 to match the fact that these are *uncertain* rules, compared to certain flow propagation and NULL dereference rules. Intuitively, higher the weight, the stronger the propagation rule. Line 7 shows an example of providing *priors*; in this case we specify that the likelihood of FLOW(x,y) is low. This will make it so that FLOW(x,y) has a high probability value only if we have direct evidence for it.

To illustrate the power of probabilistic reasoning, we run Alchemy interference on a small program, whose flow graph is shown below.




```
ERROR(W3)   0.616988
ERROR(Z3)   0.614989
ERROR(W4)   0.567993
ERROR(W8)   0.560994
ERROR(W11)  0.544996
```

Nodes represent variables and edges represent assignments (ASSIGN). Dotted edges represent conditional assignments (ASSIGNCOND). Here, red nodes for variables X1 and U1 are sources of NULL (ISNULL). Nodes in blue are dereference nodes (DEREF), which are potential locations of errors. The green node for Y1 is an object allocation node (NEWOBJ).

Analysis results are shown next to the graph. We see that W3 and Z3 are more certain than W4 because here is a conditional edge from U1 to W4. Also, as expected, the probability goes down as we traverse further down the path from W3 to W8 to W11.

This example shows the power of probabilistic analysis to prioritize errors depending on how they are inferred.

5. Challenges

We see the following challenges taking these ideas forward. First, the effectiveness of the probabilistic inference system relies on the weights assigned to each of the rules. We can imagine an expert user to provide a set of initial weights. We can also try to leverage the *learning* facilities in tools such as Alchemy to learn the weights, or improve the initial estimates. However, that would necessitate the presence of *labeled training datasets* for the underlying domain. Second, one broad question is to investigate is the class of static analysis techniques that are naturally amenable to an elastic analysis. It would be beneficial to leverage the powers of existing static analysis based on symbolic techniques.