Evaluating the Imprecision of Static Analysis

Atanas Rountev Scott Kagan Michael Gibas
Department of Computer Science and Engineering
Ohio State University
{rountev,kagan,gibas}@cis.ohio-state.edu

ABSTRACT

This work discusses two non-traditional approaches for evaluating the imprecision of static analysis. The approaches are based on proofs of feasibility or infeasibility that are constructed manually by the experimenters. We also describe our initial experience with these techniques.

Categories and Subject Descriptors

F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*program analysis*

General Terms

Experimentation, measurement, languages, algorithms

Keywords

Static analysis, dynamic analysis, analysis precision

1. INTRODUCTION

An important issue for designers of static analyses is the problem of *analysis imprecision.* Infeasibility in the analysis solution can have serious implications in the context of software tools. In software understanding, analysis imprecision may require a tool user to perform time-consuming manual examination of the code. For example, if a programmer employs a tool to assist with maintenance changes, potential problems reported by the tool may have to be investigated carefully. If many problems are "false alarms" due to imprecision in the underlying static analyses, the usefulness of the tool becomes questionable. Similarly, if analysis results are used to compute testing requirements, analysis imprecision may result in requirements that are impossible to satisfy. This makes the interpretation of coverage results harder, leading to questions such as "The coverage is only 30%; is this because the tests are inadequate, or because the coverage requirements are infeasible?". If a tester attempts to achieve higher coverage, she may have to examine the code to determine which requirements are feasible and which ones are not, essentially "fixing by hand" the analysis results at the expense of wasted time and effort.

Analysis imprecision is certainly not the only factor that affects the usefulness of software tools. However, for program analysis research, imprecision is one of the key issues that must be addressed by developing new algorithms and by performing empirical studies. We believe that the agenda for static analysis research should include the goal of achieving very low imprecision while at the same time remaining practical. In this context, one key question is how to evaluate the imprecision in existing and new static analyses. While theoretical arguments are valuable for establishing the fundamental relationships between analysis algorithms, they are far from sufficient. Thus, program analysis researchers use *empirical studies* to evaluate the imprecision of static analyses. Two traditional approaches used in such studies provide lower bounds and upper bounds on the analysis imprecision, as discussed shortly. In this paper we discuss two alternative approaches for empirical evaluation of analysis imprecision. The first approach obtains *precise* measurements of imprecision. The second one provides lower bounds on the imprecision of *entire categories* of analyses, rather than for a few specific analyses. These ideas are very much work in progress, and we hope to get feedback and to stimulate a discussion on this topic in the program analysis community.

2. TRADITIONAL EVALUATIONS

This section briefly outlines two traditional approaches for evaluating the imprecision of static analysis. First, consider some basic definitions. Let c be a software component that will be analyzed, and suppose we are interested in evaluating the imprecision of an entire family of static analyses $\{A_1, A_2, \ldots\}$ on c. Typically, such families consist of analyses that compute similar kinds of information. Therefore, without loss of generality, we will assume that the solutions $S_1(c), S_2(c), \ldots$ produced by these analyses for c are all subsets of some universal set $U(c)$ of analysis facts for c.

Let $S^*(c) \subseteq U(c)$ denote the set of *all and only* analysis facts for c that are feasible at run time. Because static analysis is intrinsically conservative, $S^*(c) \subseteq S_i(c)$ for all A_i in the analysis family under consideration. For any A_i, the *imprecision* of A_i for c is the set $\Delta_i(c) = S_i(c) - S^*(c)$.

Suppose we are performing an empirical study, and we are interested in obtaining certain measurements related to $\Delta_i(c)$ (e.g., its size) for each analysis from the family. The traditional approach for evaluating analysis imprecision can be summarized as follows: for each A_i, a *lower bound* on the imprecision is obtained as

PASTE'04, June 7-8, 2004, Washington, DC, USA.

$$S_i(c) - \bigcap_j S_j(c) \;\subseteq\; \Delta_i(c)$$

In other words, every analysis fact in $S_i(c)$ which is not in all solutions is definitely an infeasible fact. Since $S^*(c)$ in general is uncomputable, this approach is often used to identify indirectly some of the elements of $\Delta_i(c)$. Of course, the quality of this lower bound depends on the analysis family. A particularly common case is when the family contains two analyses A_1 and A_2, and A_1 is provably at least as precise as A_2. In this case $S_1(c) \subseteq S_2(c)$ and therefore $S_2(c) - S_1(c) \subseteq \Delta_2(c)$.

A disadvantage of such lower bounds is that it is unclear how tight they are—that is, how close they are to the actual imprecision. Some researchers have addressed this problem by using an alternative approach for evaluating analysis imprecision. This is achieved by executing the component several times. During these executions, a dynamic analysis observes the run-time behavior and determines the subset $S'_i(c)$ of the static solution $S_i(c)$ which occurred at run time. This provides an *upper bound* on the imprecision, since $\Delta_i(c) \subseteq S_i(c) - S'_i(c)$.

The combination of these two approaches defines an interval for the imprecision of a static analysis. If this interval is small, it provides valuable insights about an analysis A_i. However, this is not always the case. To get a tight lower bound, we need an analysis family that contains at least one very precise analysis. The quality of the upper bounds depends on the run-time executions under consideration. Sometimes claims are made that these upper bounds are probably close to the actual imprecision, but there is usually no evidence to support such claims. When the two bounds are relatively far from each other, neither one provides a good indication of the degree of analysis imprecision.

3. PRECISE EVALUATIONS

This section describes an approach for imprecision evaluation that can be thought of as the second technique from above "taken to the extreme". The idea is simple: the evaluation considers the difference between $S_i(c)$ and the run-time subset $S'_i(c)$. For each element of this difference, either run-time coverage is achieved by extending the set of run-time executions (e.g., by adding new test inputs), or a proof is constructed that no run-time execution could ever cover the element. In other words, for each $e \in S_i(c) - S'_i(c)$, the experimenters either construct a proof of e's feasibility (proof by existence, obtained by observing the run-time behavior of c), or a proof of infeasibility (which must be done on paper using human intelligence). The experimenters can be the researchers themselves, or people that are performing the study under the supervision of the researchers. The outcome of the study is the exact set $\Delta_i(c)$.

Such methodology immediately raises the following question: how can the experimenters be sure that the classification of e as feasible or infeasible is correct? While proofs of feasibility are self-evident ("since it was observed at run time, it is feasible"), proofs of infeasibility are less obvious. Thus, it is possible that the experimenters constructed incorrect proofs of infeasibility, and this compromised the results of the study. We believe that despite this issue, the methodology is perfectly viable if applied carefully. As an example, consider the role of proofs in mathematics. Given a proof constructed by one mathematician, the correctness of this proof is typically checked manually by other mathematicians. While human errors are possible (and do happen), such manual proof verification has been used successfully for centuries. When evaluating static analysis imprecision, proofs of infeasibility can be constructed by one experimenter and then checked by one or more other experimenters. An even stronger version of this approach is to ask several experimenters to construct infeasibility proofs independently from each other, and subsequently to check each other's proofs. This second version was used by two authors of the paper in the study described in Section 5.

Another potential disadvantage of this approach is that it may be labor-intensive. It may take many studies performed by different research groups before a sufficient body of evidence is collected to draw strong conclusions about the imprecision of different analyses. However, this is inevitable in any experimental science: long-term gathering and replication of experimental data by a research community is the standard methodology in many other scientific disciplines. One may also argue that the effort invested by the researchers to *evaluate* their analysis should be at least as significant as the effort required by an analysis client to *use* this analysis. If an analysis client (e.g., a maintenance programmer or a tester) has to spend time "figuring out" the infeasibility of the analysis solution, shouldn't the researchers be required to do the same?

4. SOURCES OF IMPRECISION

After determining precisely $\Delta_i(c)$, additional insights can be gained about the effects of different sources of imprecision in A_i. The idea is to consider the different approximations that are made by the analysis algorithm. Examples of commonly-used approximations are: (1) assuming that all paths in a procedure's control-flow graph are feasible; (2) ignoring the intraprocedural flow of control, as done by flow-insensitive analyses; (3) ignoring the calling context of procedures, as done by context-insensitive analyses. Each such approximation is a potential source of imprecision, and may contribute to some portion of $\Delta_i(c)$.

For each kind of approximation, its impact on $\Delta_i(c)$ can be evaluated using the following approach. First, it is necessary to define an *abstracted semantics* for the analyzed language which reflects the effect of the approximation. For example, suppose we consider the standard approach of imprecise modeling for conditions in `if`, `switch`, `while`, etc. In this approach, for a statement `if(p*q>5)..`, an analysis does not attempt to track the values of `p` and `q` and to decide whether the condition is true or false; rather, the actual condition is abstracted away and both the true and the false outcome are considered possible. This approximation can be modeled by defining an abstracted semantics which differs from the standard semantics *only* in its treatment of conditions. In the abstracted semantics, each run-time evaluation of a condition results in a random choice between true and false. Compared with the standard semantics, the only difference is the choice of the next statement to be executed: after evaluating a condition, rather than using the resulting boolean value, the execution randomly chooses one of the two possible next statements. The semantics of `switch` and `while` statements can be modified in a similar fashion.

In this model, there may be multiple possible valid executions for c on the same input. If $e \in \Delta_i(c)$ occurs during one of these valid executions, this means that *any* static analysis which abstracts away conditions *must* report e as feasible. This theoretical model allows us to draw conclusions about all possible static analyses that employ this approximation. Thus, if e is infeasible in the standard semantics but is feasible in the abstracted semantics, it will be erroneously reported as feasible by any such analysis.

For each $e \in \Delta_i(c)$, the experimenters can construct proofs of feasibility or infeasibility in the abstracted semantics. The portion of $\Delta_i(c)$ that is feasible in the abstracted semantics allows us to quantify precisely the effect of the approximation on analysis imprecision. For example, suppose that 40% of the elements of $\Delta_i(c)$ are proven feasible in the abstracted semantics. This means that regardless of how much one improves all other aspects of the algorithm, as long as conditions are abstracted, at least 40% of the imprecision will still remain in the analysis solution.

For each approximation, a different abstracted semantics can be defined. For example, to model intraprocedural flow insensitivity, the abstracted semantics should choose randomly the next statement to be executed among all statements in the procedure. It remains an open problem to define similar formalisms for other common approximations. Intuitively, it seems that it should be easy to achieve this goal: after all, one of the standard views of static analysis (abstract interpretation [1]) is as execution in some abstracted semantics.

5. PRELIMINARY EXPERIENCE

In this section we briefly summarize a study which used the two evaluation techniques outlined above; the details of the experiments are available in [3]. The study evaluated the precision of two class analyses for the purpose of computing call chains in Java components. *Class analysis* computes an overestimate of the classes of all objects to which a given reference variable may point. There is a large body of work on class analysis, most of which is summarized in [2, 4]. One of the uses of class analysis is to compute call graphs. A *call chain* is a sequence of call graph edges $e_1, \ldots, e_k$ such that the target of e_i is the same as the source of e_{i+1} for $1 \leq i < k$. Call chains have a variety of uses for program understanding and testing [3], and one goal of the study was to determine how many of the call chains in a call graph were actually feasible. More precisely, we considered all call chains of certain length in all components in the study.

The class analyses used in the experiments were Rapid Type Analysis (RTA) and Andersen's analysis (AND); both are well-known class analyses. Let Δ_{RTA} and Δ_{AND} be the sets of infeasible call chains reported by the two analyses. AND is more precise than RTA, and this provides a lower bound for Δ_{RTA}. We started the study with a set of test cases from the Mauve open-source test suite for the standard Java libraries (`sources.redhat.com/mauve`). The run-time coverage for these tests was used to obtain upper bounds on Δ_{RTA} and Δ_{AND}. Using the two upper bounds and the one lower bound, we concluded that (1) between 4% and 51% of the call chains reported by RTA were infeasible, and (2) at most 49% of the call chains reported by AND were infeasible. Clearly, these results were not particularly informative.

We then performed experiments to determine precisely Δ_{RTA} and Δ_{AND}. As a result, we discovered that 26% of the call chains reported by RTA were infeasible, and 22% of the chains for AND were infeasible. This provided us with much better understanding of the degree of imprecision in both analyses. To obtain these results, we had to prove/disprove the feasibility of 445 call chains that were not covered at run time. This effort was about one person-month, which was quite reasonable. Two of the authors worked independently from each other to constructs proofs of feasibility or infeasibility for each investigated call chain. These proofs were then cross-checked to ensure correctness.

Note that these results illustrate the danger of using dynamic analysis to obtain upper bounds on imprecision: since the Mauve tests did not achieve high run-time coverage, the bounds implied by them were quite different from the actual imprecision (e.g. 49% vs. 22% for AND).

We also evaluated the impact of one particular source of analysis imprecision: the standard approach of imprecise modeling for conditions in `if`, `switch`, `while`, etc. The same two authors constructed proofs of feasibility or infeasibility in the appropriate abstracted semantics. The effort was relatively low (about one person-week), partly because these experiments were performed after the investigation of Δ_{RTA} and Δ_{AND} and they benefited from the gained familiarity with the subject components. The results revealed that 95% of the infeasible chains were due to the imprecise handling of conditions. Since this is a common approximation for all class analyses, these experiments showed that more powerful class analyses would have resulted in very little (if any) precision improvement.

6. CONCLUSIONS AND FUTURE WORK

Our preliminary study indicates that valuable insights can be gained by the two evaluation techniques proposed in this paper. First, better understanding of the degree of imprecision can be obtained: e.g. "the imprecision is 22%" rather than "the imprecision is at most 49%". Such insights allow analysis designers to decide how important it is to focus more effort on reducing analysis imprecision. Furthermore, quantifying the sources of imprecision (e.g., "95% of the imprecision is due to the handling of conditions") provides guidance for future work on analysis improvement.

More studies of this nature need to be carried out by various research groups in order to gain more experience with this methodology. One obvious concern is the human effort required to perform the experiments. The analysis problems for which this approach is practical remain to be determined. We believe that the labor costs for an individual research group could be reasonable: for example, even undergraduate students could participate in the experiments. The cumulative effect of such studies can be a large body of data which conclusively resolves open questions about analysis imprecision. Organizing public repositories of software components together with the sets $S^*(c)$ of feasible analysis facts can be the first step in this process.

7. REFERENCES

[1] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixed points. In *POPL*, 1977.

[2] D. Grove and C. Chambers. A framework for call graph construction algorithms. *TOPLAS*, 23(6):685–746, 2001.

[3] A. Rountev, S. Kagan, and M. Gibas. Static and dynamic analysis of call chains in Java. In *ISSTA*, 2004.

[4] B. G. Ryder. Dimensions of precision in reference analysis of object-oriented programming languages. In *CC*, 2003.