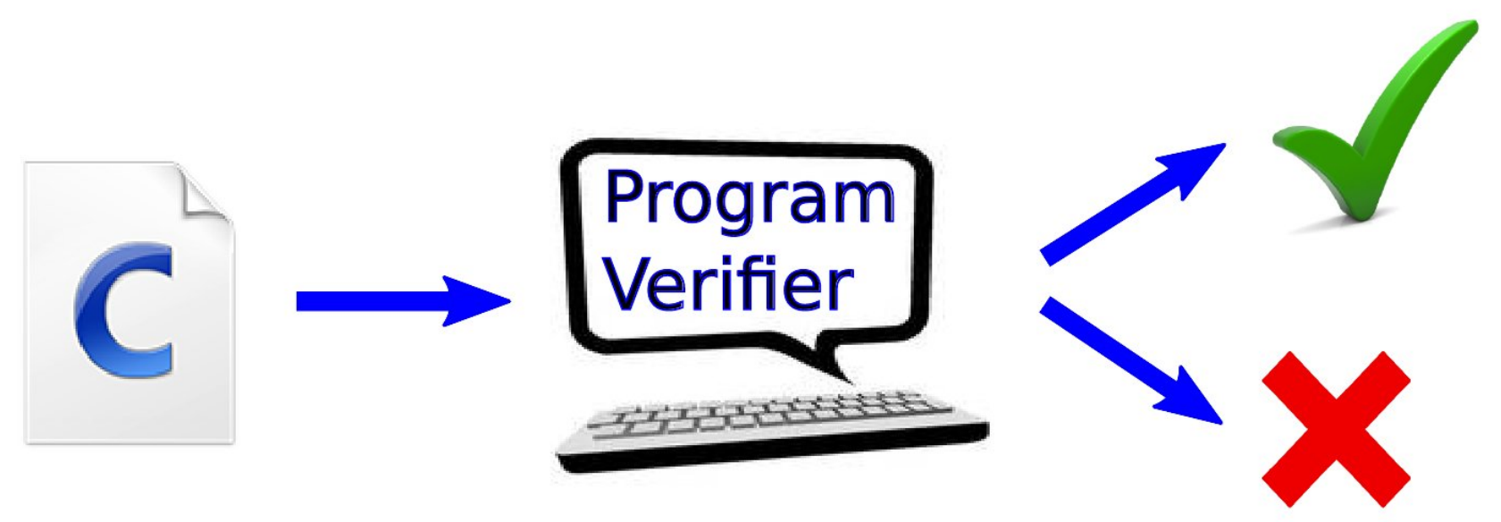
Automated Error Diagnosis Using Abductive Inference

Işıl Dillig
College of William & Mary

Thomas Dillig
College of William & Mary

Alex Aiken
Stanford University

Motivation



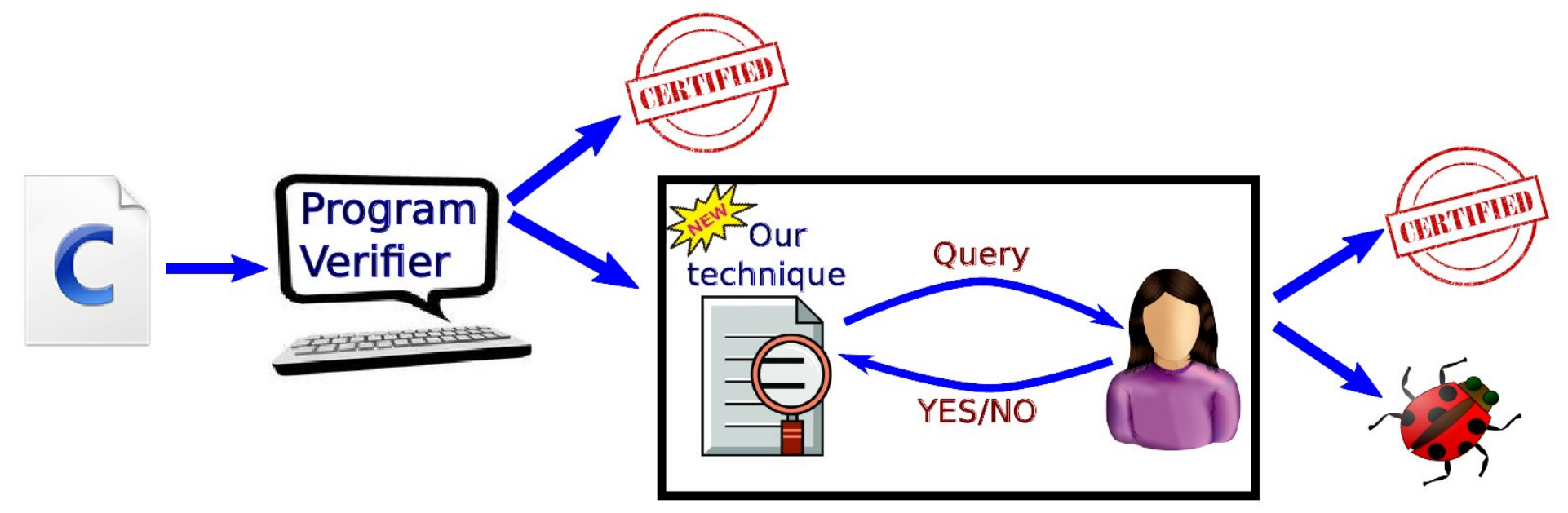
- If we use sound program analysis tool to verify a property, answer is either yes or no
- If answer is yes, program is error-free
- If answer is no, there are two possibilities:
  - Either the program is indeed buggy
  - Or report is a false alarm

When Verification Fails

- When verifier fails to prove property, user must decide whether report is real bug or false alarm.
- But manually classifying error reports is time-consuming and error-prone.
- Furthermore, user must redo all the reasoning the tool performed just to discover where it became stuck.
- Very painful process for most users of static analysis tools!

Our Goal

A new technique for semi-automating error report classification when automated program verification fails



- Allows verifier to interact with user by asking small, relevant queries until report is classified as real bug or false positive
- Queries capture only the information verifier is missing ⇒ user contributes facts verifier could not decide on its own
- Answering queries much easier than classifying error report

Key Ideas

Key Idea #1: Analysis makes explicit not only facts it knows, but also facts it does not know

- Sources of imprecision/incompleteness in static analysis represented using abstraction variables
- For example, if value of variable is unknown after a loop, represent this unknown value using abstraction variable
- This representation allows analysis to be "introspective" and reason about what facts it could be missing

Key Ideas, cont.

Key Idea #2: Abductive inference

- Given known facts F and desired outcome O, abductive inference finds simple explanatory hypothesis E such that

$$F \wedge E \models O \quad \text{and} \quad \mathrm{SAT}(F \wedge E)$$

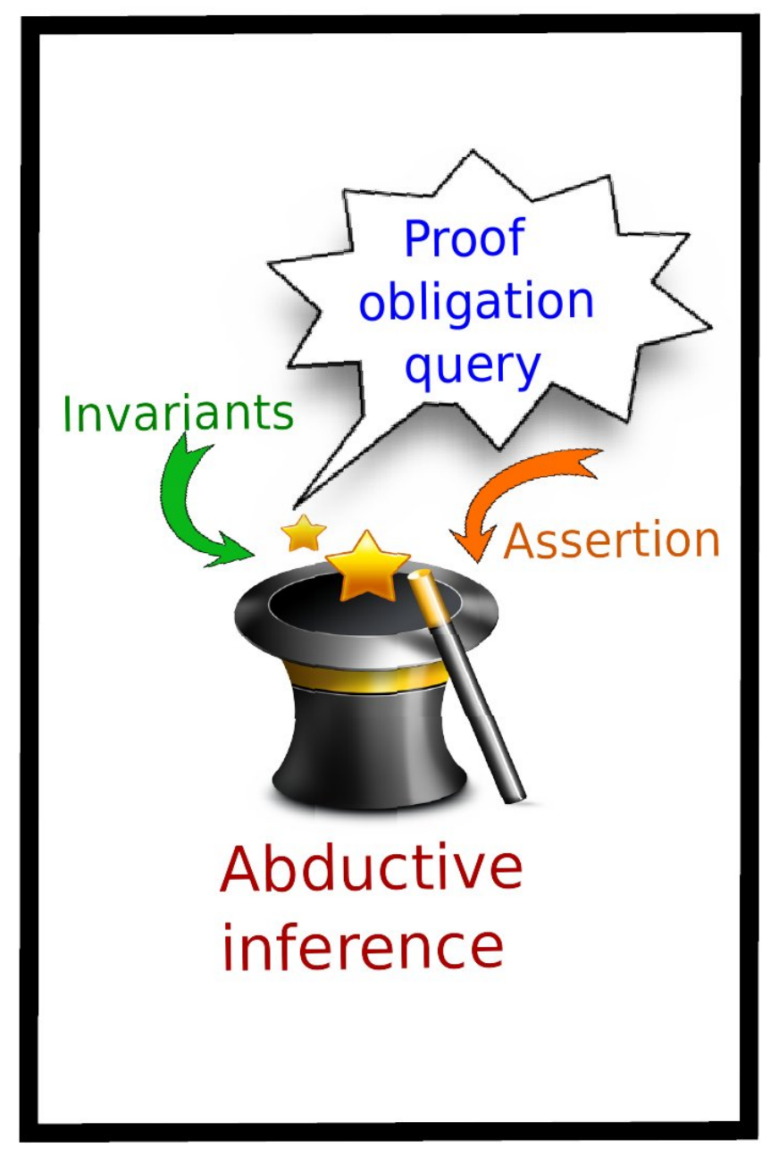
- We use abductive inference to generate simple explanations that either guarantee that program is error-free or definitely buggy
- These abductive explanations are presented as queries to user

Proof Obligation via Abductive Inference

- Input: invariants computed by verifier and assertion to discharge
- Technique computes formulas I and ϕ describing invariant and assertion in terms of abstraction variables
- Use abduction to compute simple and general explanation Γ s.t.:

$$\Gamma \wedge I \models \phi \quad \text{and} \quad \mathrm{SAT}(\Gamma \wedge I)$$

- Abductive explanation Γ is presented to user as proof obligation query
- If Γ is invariant, report is false alarm

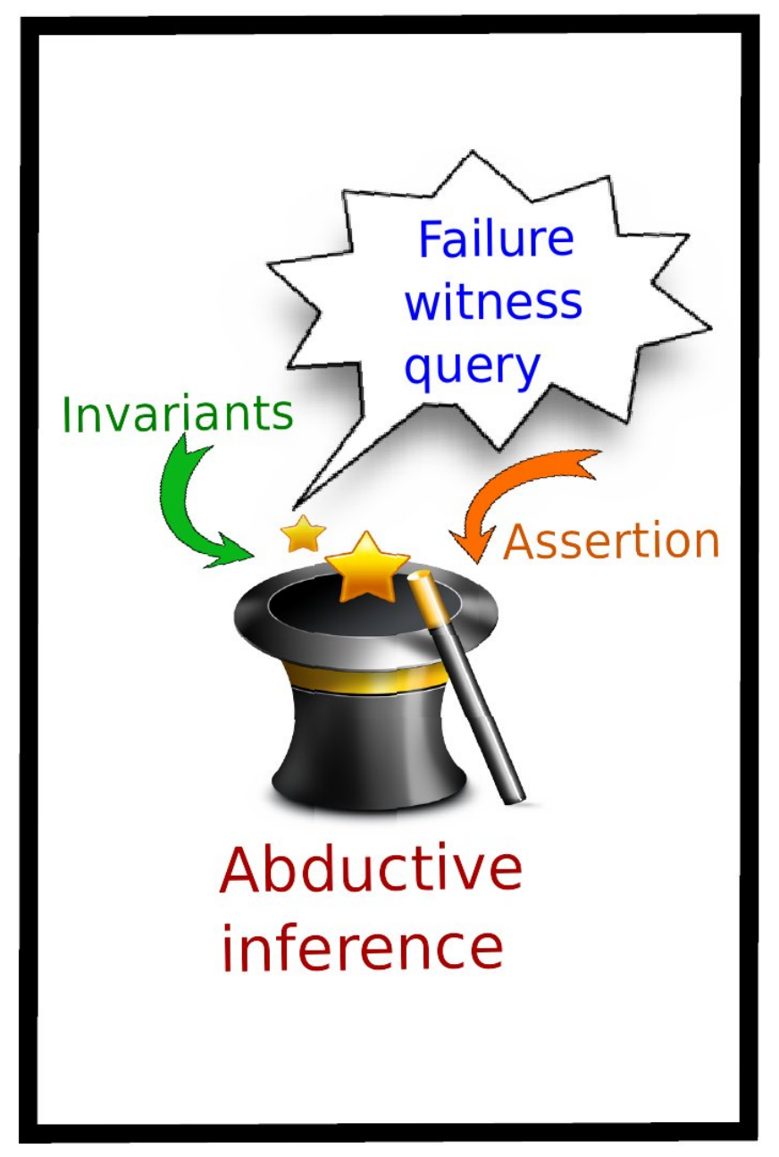


Failure Witnesses

- Proof obligation query used to show report is false alarm
- We generate another query, called failure witness query, to show report is a real bug
- To generare failure witness query, solve a dual abductive inference problem:

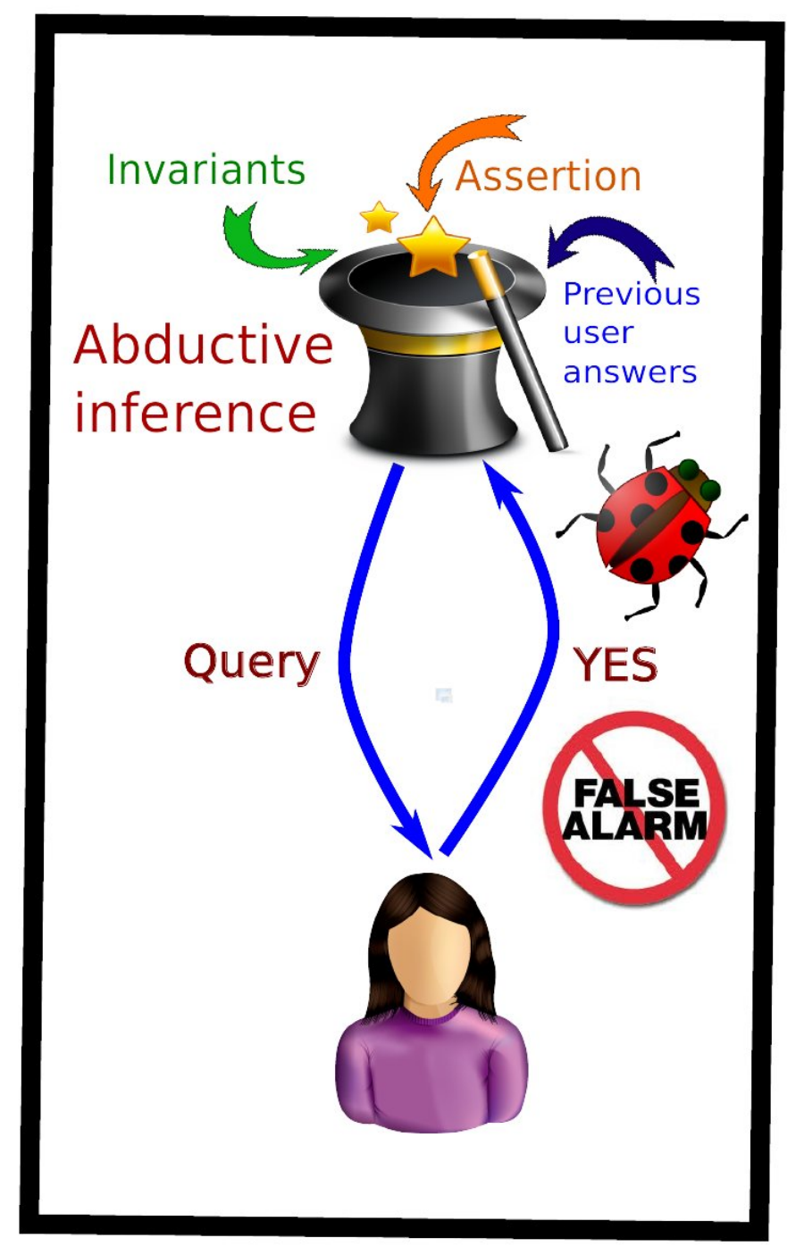
$$\Delta \wedge I \models \neg\phi \quad \text{and} \quad \mathrm{SAT}(\Delta \wedge I)$$

- If Δ can hold in some program execution, then report is real bug!



Automated Error Diagnosis via Abductive Inference

- Our technique helps user classify error reports by generating simple queries
- If query is a proof obligation and user answers yes, report classified as false alarm
- If query is a failure witness and user answers yes, report classified as real bug
- If user answers "no" or "I don't know", technique computes new abductive explanation distinct from previous ones
- Interaction continues until report is classified as real bug or false alarm

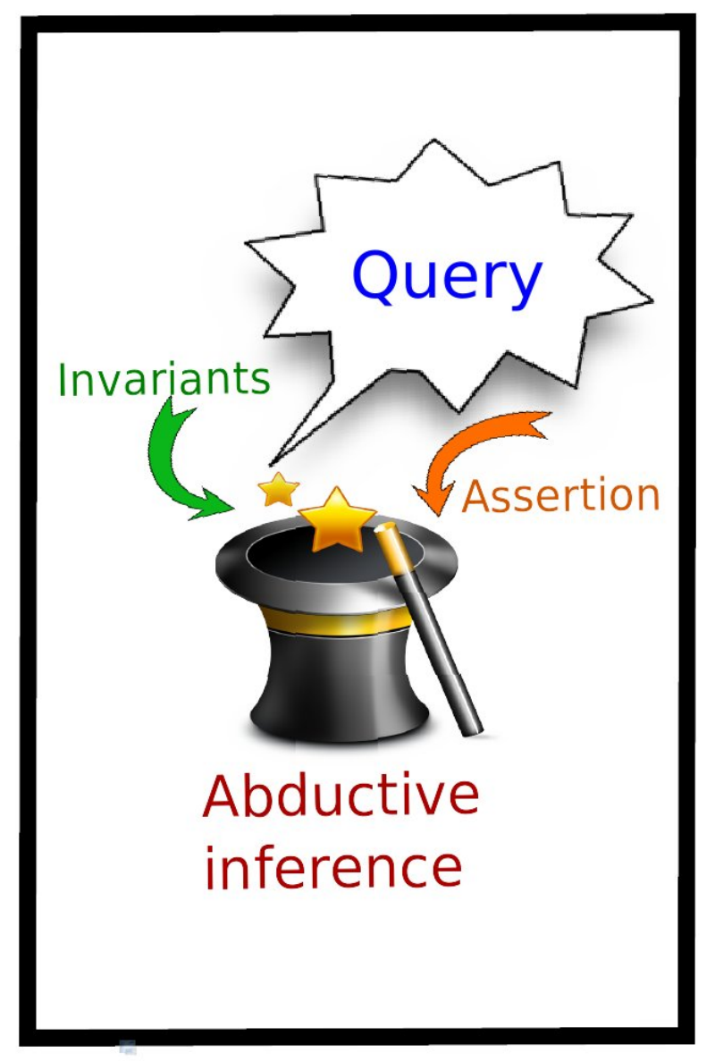


Computing Abductive Explanations

- Abduction is useful, but how do we compute these explanations?
- Given invariants I and desired outcome ϕ, how to find explanation E s.t.:

$$I \wedge E \models \phi \quad \wedge \quad \mathrm{SAT}(I \wedge E)$$

- Trivial solution is $E = \phi$, but useless b/c same as asking user to prove assertion!
- Want solutions that are as simple and as general as possible!



Use minimum satisfying assignments and quantifier elimination to compute simple and general explanations

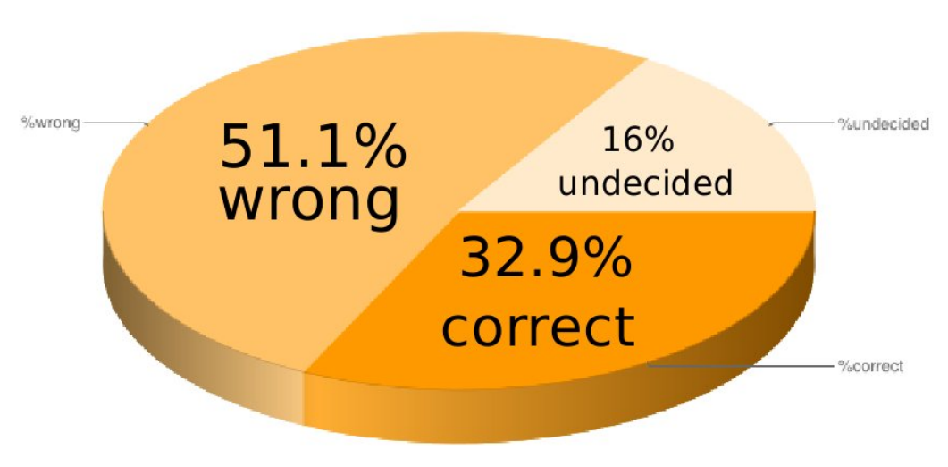
Experimental Evaluation

- Performed user study to evaluate new technique
- Hired 56 programmers through ODesk and asked them to classify error reports
- Each programmer asked to classify (randomly selected) half of reports manually, and other half using our technique
- Manual classification: Given code and error report, decide if bug, false alarm, or unknown
- Our technique: Given code and series of queries, asked to answer "Yes", "No", or "Don't know"
- Based on answers to queries, report classified automatically

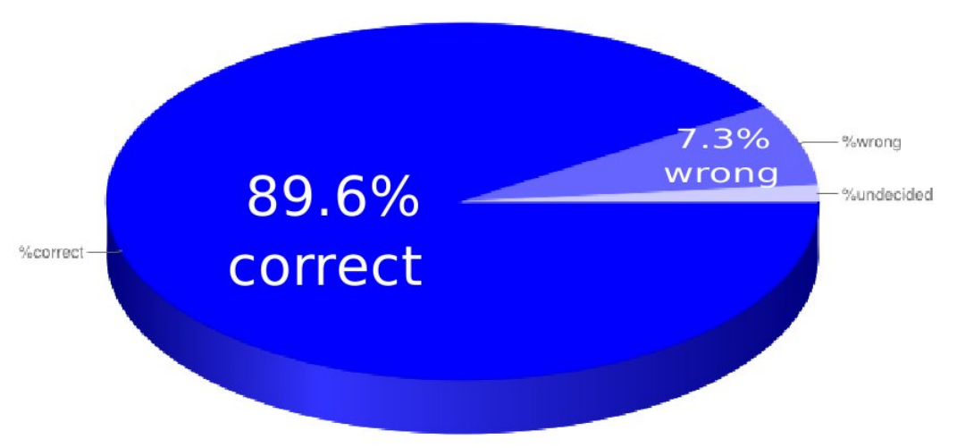
Results of User Study

- With manual classification, programmers classified 51.1% of reports incorrectly
- With assisted classification, programmers classified only 7.3% of reports incorrectly
- Our technique dramatically improves classification accuracy
- Also dramatically reduces time needed to classify report
- Using manual classification, programmers need 293 seconds on average
- Using new technique, programmers take 55 seconds on average

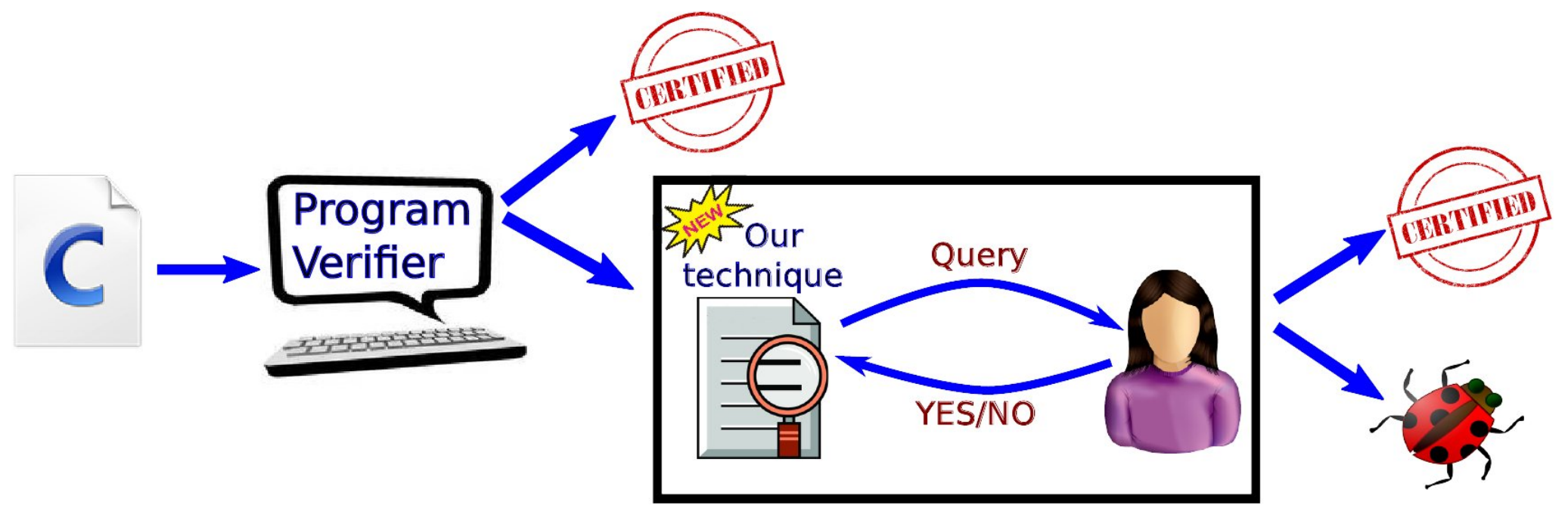
Manual Classification



Assisted Classification



Summary



- New technique to help programmers classify error reports as real bugs or false alarms
- Uses abductive inference to compute simple queries that capture what analysis is missing
- Interacts with user until report is classified as bug/false alarm