Infeasible Path Generalization in Dynamic Symbolic Execution

Micka¨el Delahayea
University of Grenoble

Bernard Botellaa
CEA · Campus d'Innovation en Micro et Nanotechnologies (MINATEC)

Arnaud Gotliebb
Simula Research Laboratory · Department of Software Engineering

Meta Data & Stats

Published in:	IST
Year:	2014
Number of Authors:	3
Citations:	18
Pages (PDF):	20
Figures:	12
References:	38
Formals:	7 definitions

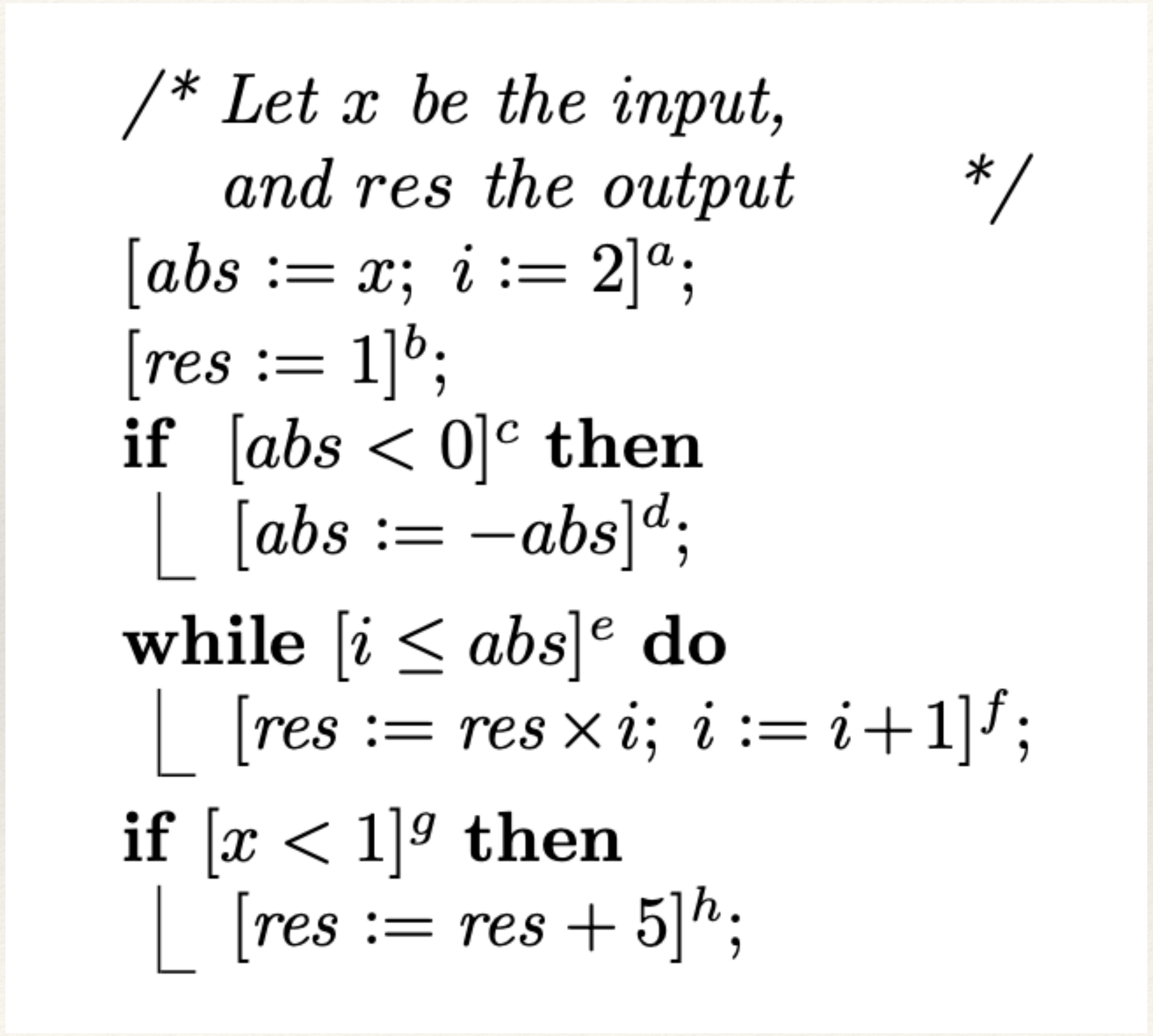
Table of Content

Introduction

Objective: When selecting paths in DSE for generating test inputs, some paths are actually detected as being infeasible, meaning that no input can be found to exercize them. But, showing path infeasibility instead of generating test inputs is costly and most effort could be saved in DSE by reusing path infeasibility information.

Method: In this paper, we propose a method that takes opportunity of the detection of a single infeasible path to generalize to a possibly infinite family of infeasible paths. The method first extracts an explanation of path condition, that is, the reason of the path infeasibility. Then, it determines conditions, using data dependency information, that paths must respect to exhibit the same infeasibility. Finally, it constructs an automaton matching the generalized infeasible paths.

Example



```
/* Let $x$ be the input,
   and $res$ the output                */
$[abs := x;\ i := 2]^a$;
$[res := 1]^b$;
if $[abs < 0]^c$ then
    $[abs := -abs]^d$;
while $[i \leq abs]^e$ do
    $[res := res \times i;\ i := i + 1]^f$;
if $[x < 1]^g$ then
    $[res := res + 5]^h$;
```

- In step (1), an input is arbitrarily chosen (e.g., x = 2), the program is executed, and the activated path is traced (a b cf e t f ef g f)

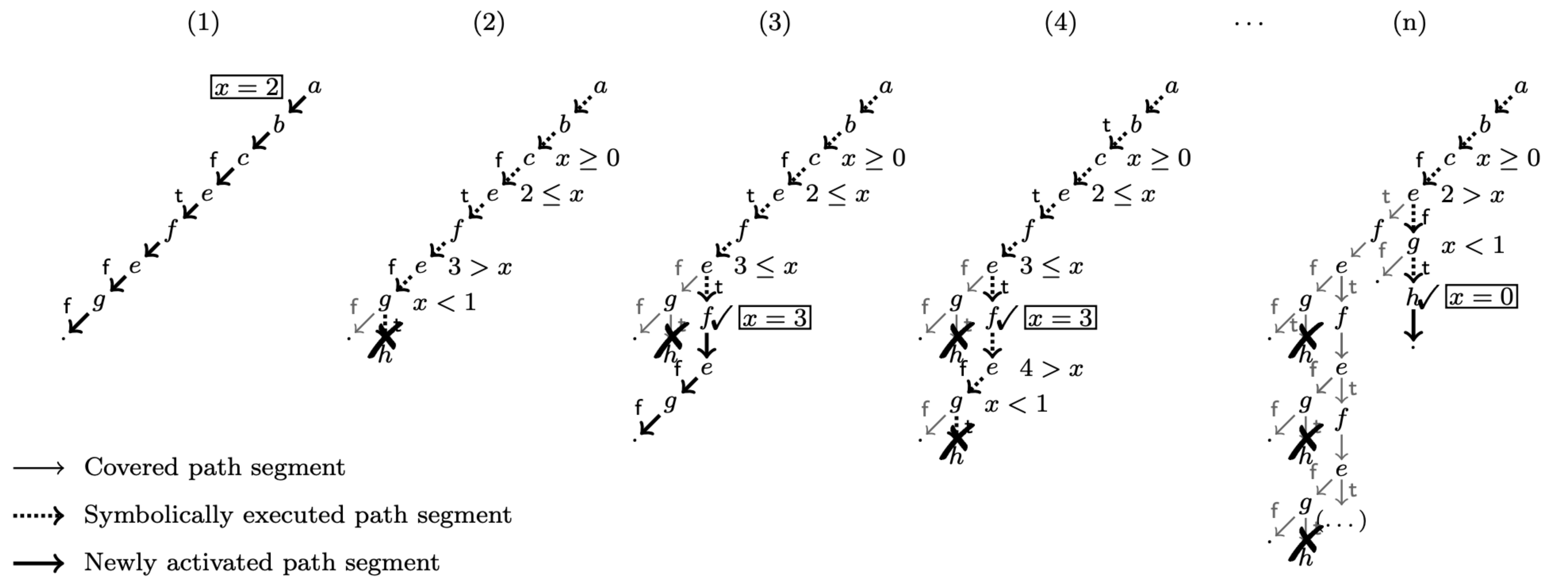
- In step (2), the tool chooses a new path to cover (indicated with dotted arrows) based on the current path using a depth-first strategy

- In (3), the tool tries to activate another path with the same method. This time, the solver gives a solution to the path condition. This solution (x = 3)

- In (4), the tool tries to activate the statement h, and for the very same reason as step (2), the attempt fails.

Example

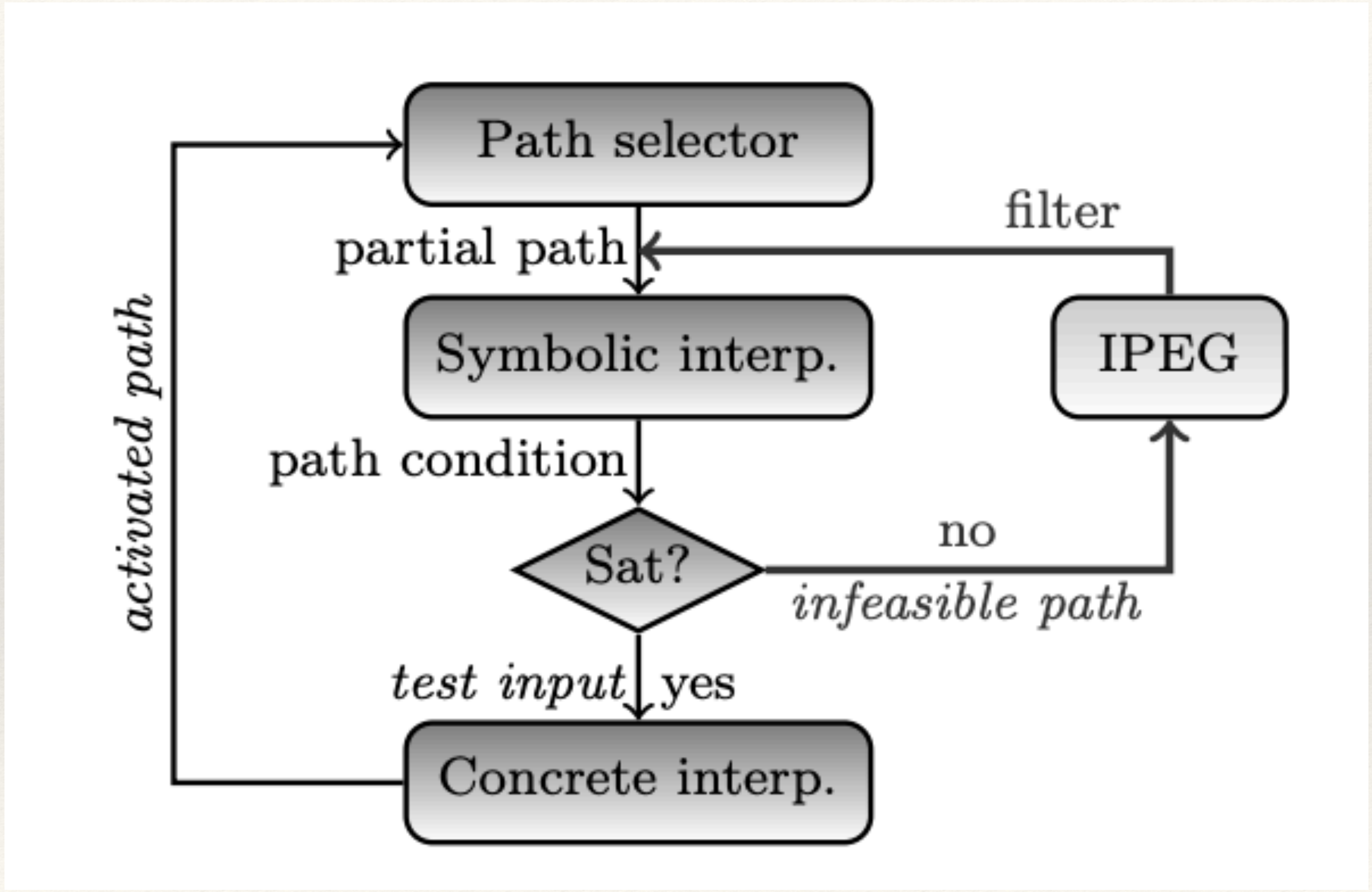
```
/* Let x be the input,
   and res the output                    */
$[abs := x;\ i := 2]^a;$
$[res := 1]^b;$
if $[abs < 0]^c$ then
    $[abs := -abs]^d;$
while $[i \leq abs]^e$ do
    $[res := res \times i;\ i := i+1]^f;$
if $[x < 1]^g$ then
    $[res := res + 5]^h;$
```



An infeasible path automaton

$$\begin{tikzcd}
{} \arrow[r] & \bullet \arrow[r, "a"] & \bullet \arrow[r, "b"] & \bullet \arrow[r, "c^{\mathsf{f}}"] & \bullet \arrow[r, "e^{\mathsf{t}}"] & \bullet \arrow[r, "f"] & \bullet \arrow[r, "e^{\mathsf{f}}"] \arrow[d, bend left, "e^{\mathsf{t}}"] & \bullet \arrow[r, "g^{\mathsf{t}}"] & \odot \\
& & & & & & \bullet \arrow[u, bend left, "f"] & &
\end{tikzcd}$$

Integration to DSE



Results

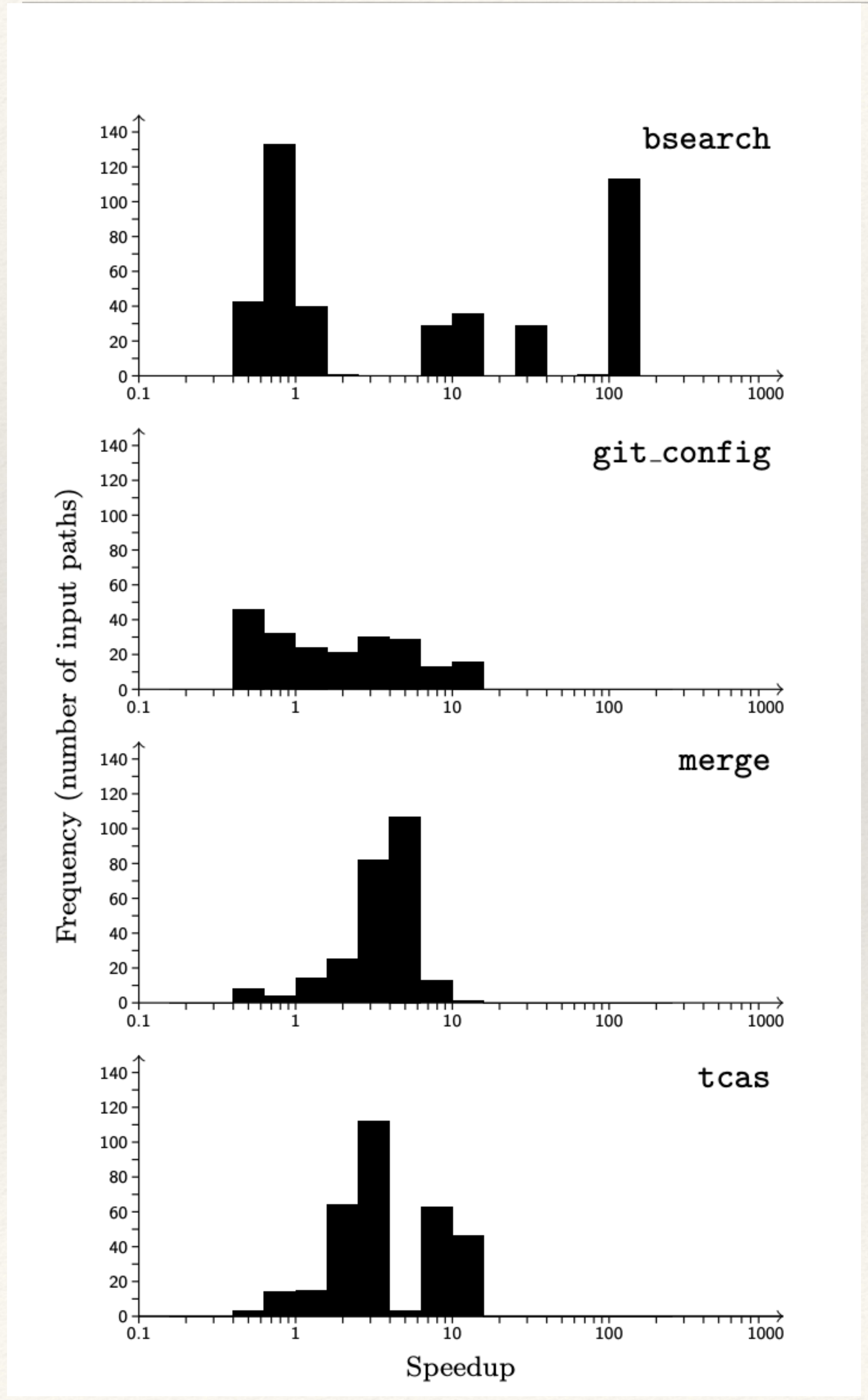


Figure illustrates by histograms the distribution of the speedup for each input infeasible path on four programs (merge, tcas, bsearch, and git config) with Z3 solver.

These histograms show how the gain can vary in function of the input infeasible paths.

$$S = \frac{T_I + T_E}{T_I + T_G}$$

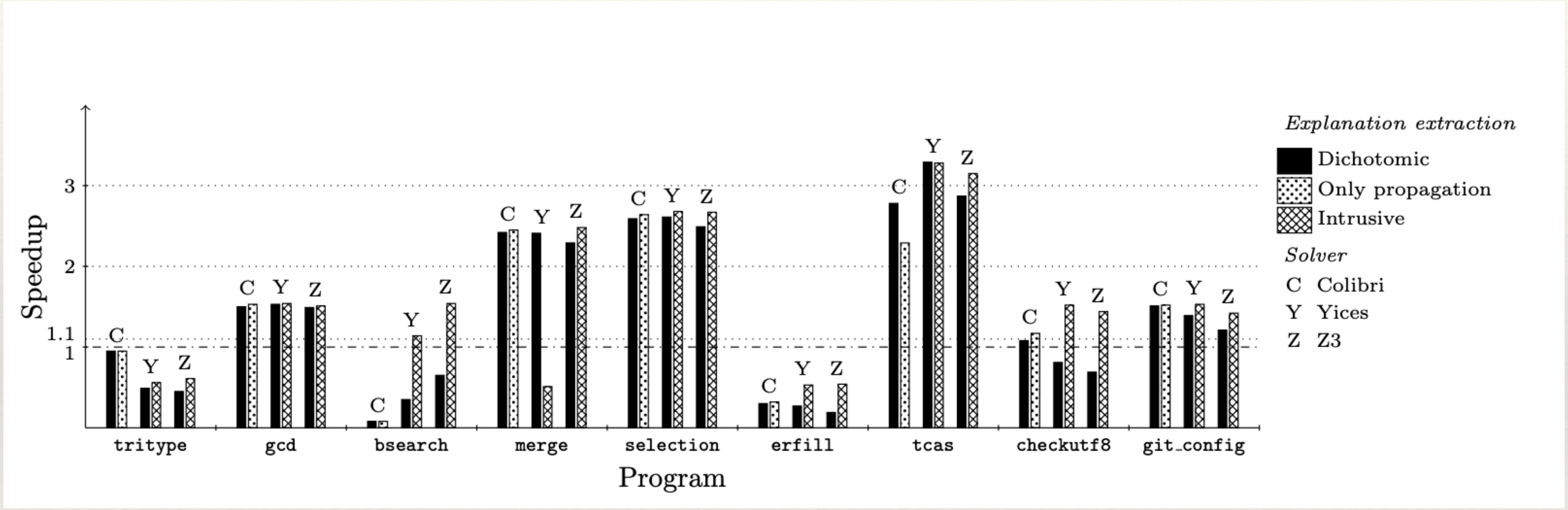
where

TI is the time needed to prove the input infeasible path infeasible,

TG the time needed for the generalization

TE the time needed to prove that every path generalized are infeasible.

Results



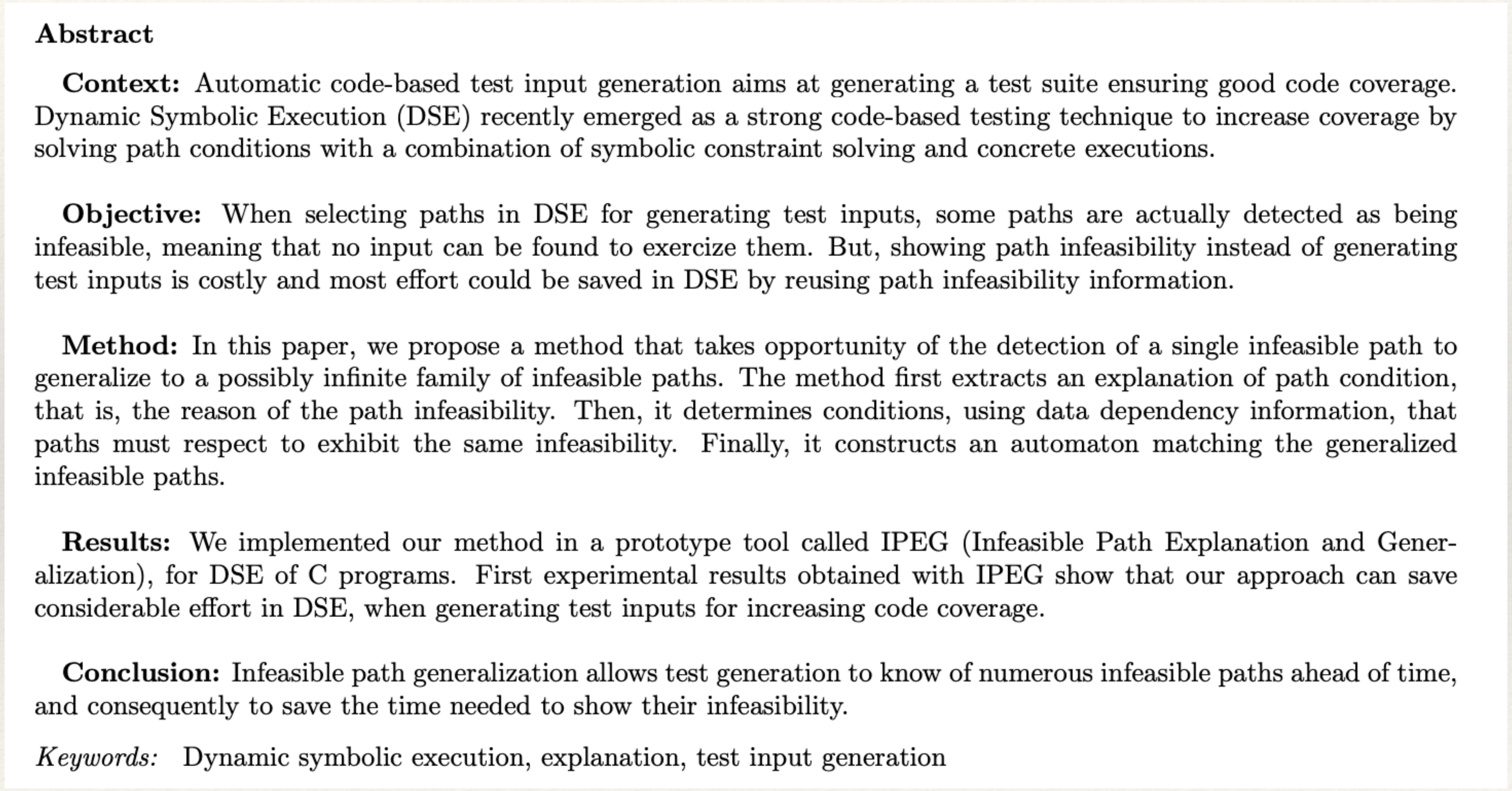
The speedup is simply T1/T2
where T1 is the time needed to generate tests without infeasible path generalization for every path of size below a certain limit
T2 the time needed to generate the same tests with generalization enabled.

Feedback

- *Problem statement(research statement is clear) +*
- *Innovation (the work brings new innovation ideas) +*
- *Contribution(IPEG) +*
- *Logical correcteness +*
- *Proof of statements +*
- *Readablity -*

What is good/interesing about the paper

- *Structured*
- *Detailed example*
- *Novel approach*



Abstract

Context: Automatic code-based test input generation aims at generating a test suite ensuring good code coverage. Dynamic Symbolic Execution (DSE) recently emerged as a strong code-based testing technique to increase coverage by solving path conditions with a combination of symbolic constraint solving and concrete executions.

Objective: When selecting paths in DSE for generating test inputs, some paths are actually detected as being infeasible, meaning that no input can be found to exercize them. But, showing path infeasibility instead of generating test inputs is costly and most effort could be saved in DSE by reusing path infeasibility information.

Method: In this paper, we propose a method that takes opportunity of the detection of a single infeasible path to generalize to a possibly infinite family of infeasible paths. The method first extracts an explanation of path condition, that is, the reason of the path infeasibility. Then, it determines conditions, using data dependency information, that paths must respect to exhibit the same infeasibility. Finally, it constructs an automaton matching the generalized infeasible paths.

Results: We implemented our method in a prototype tool called IPEG (Infeasible Path Explanation and Generalization), for DSE of C programs. First experimental results obtained with IPEG show that our approach can save considerable effort in DSE, when generating test inputs for increasing code coverage.

Conclusion: Infeasible path generalization allows test generation to know of numerous infeasible paths ahead of time, and consequently to save the time needed to show their infeasibility.

Keywords: Dynamic symbolic execution, explanation, test input generation

What could be better

- *Reability*

8. Conclusion