HIERARCHICAL DELTA DEBUGGING (HDD): SIMPLIFYING FAILURE ISOLATION IN STRUCTURED INPUTS

"REVOLUTIONIZING DEBUGGING BY HARNESSING HIERARCHY: FASTER, SMARTER, AND SIMPLER FAILURE ISOLATION WITH HIERARCHICAL DELTA DEBUGGING."

PAPER AUTHORS: GHASSAN MISHERGHI, ZHENDONG SU
INSTITUTION: UNIVERSITY OF CALIFORNIA, DAVIS

PAPER PRESENTATION BY PRIYANSH BHUVA
40269498

INTRODUCTION

- *What is Debugging? The process of identifying and fixing bugs in software.*
- *Challenge: Failure-inducing inputs are often large and complex, making manual debugging slow and difficult.*
- *Solution: Delta Debugging (DD) automates input minimization but struggles with structured inputs.*
- *Objective of Paper: Introduce Hierarchical Delta Debugging (HDD) to improve efficiency and output quality for structured inputs like XML and source code.*

File	size (tokens)	bug report (id)	ddmin tests (# of tests)	HDD tests (# of tests)
bug.c	277	unknown [20]	680	86
boom7.c	420	663	3727	144
cache.c	25011	1060	1743	191
cache-min.c	145	1060	1074	114

MOTIVATION

- Time-Consuming Debugging: Programmers spend more time debugging than any other activity.
- Inefficiency of Flat Delta Debugging: Treats inputs as flat lists, ignoring hierarchical structures.
- Real-World Problems: Large test cases from GCC and Mozilla contain irrelevant data that complicates debugging.
- Need for Structured Approach: Leveraging hierarchical structures can prune irrelevant sections early, speeding up the debugging process.

```
double mult(double *z, int n)
{
  int i;
  int j;
  for (j=0;j<n;j++) {
    i=i+j+1;
    z[i]=z[i]*(z[0]+0);
  }
  return z[n];
}
```

(a) The minimized first level.

```
mult(double *z, int n)
{
  int i;
  int j;
  for (j=0;j<n;j++) {
    i=i+j+1;
    z[i]=z[i]*(z[0]+0);
  }
}
```

(b) The minimized second level.

```
mult(double *z, int n)
{
  int i;
  int j;
  for (;;) {
    i=i+j+1;
    z[i]=z[i]*(z[0]+0);
  }
}
```

(c) The final output.

HIERARCHICAL DELTA DEBUGGING

The core technique is an enhancement of the original Delta Debugging algorithm. Instead of treating inputs as flat sequences, HDD applies Delta Debugging at each level of a hierarchical input structure (like an Abstract Syntax Tree for source code). The algorithm starts from the coarsest level and progressively refines the input, minimizing failure-inducing configurations at each level. This hierarchical approach significantly reduces irrelevant portions early in the process.

Steps:

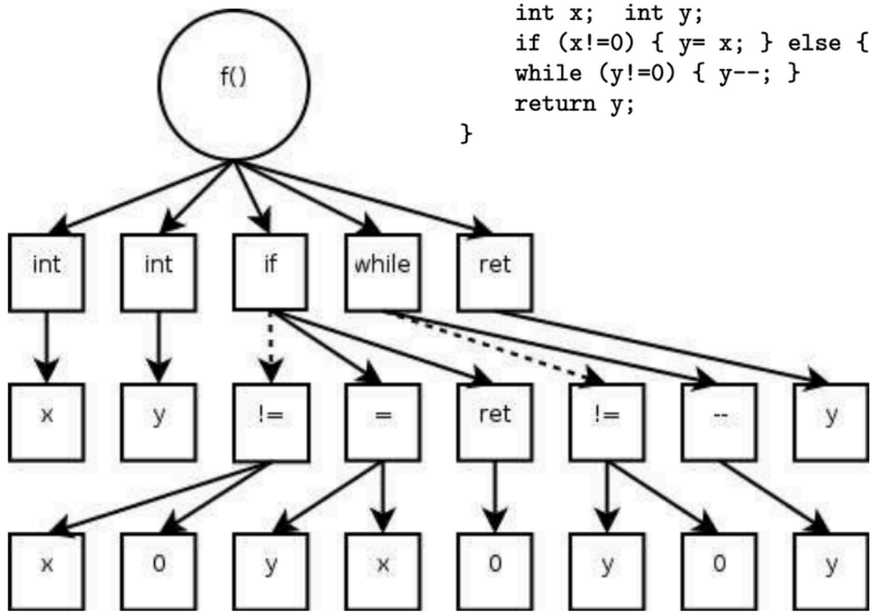
- Parse the failure-inducing input into a hierarchical structure (e.g., an AST).
- Apply Delta Debugging at the top level of the hierarchy.
- Recursively apply the algorithm to deeper levels until the input is minimized.
- Ensure syntactic validity by using context-free grammars to prevent the generation of invalid test cases.

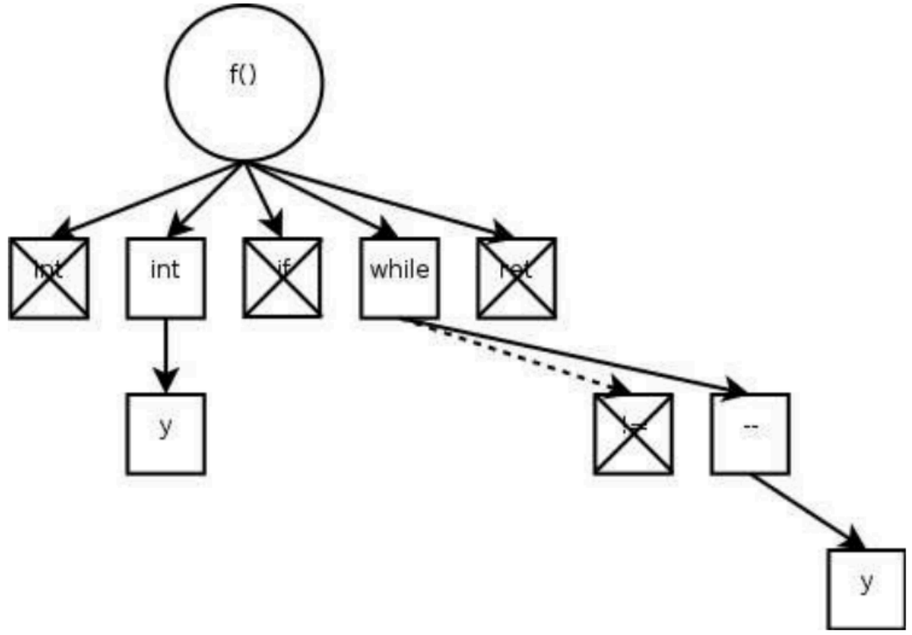
Result: Faster minimization, simpler outputs, and fewer inconclusive tests.

MAIN CONTRIBUTIONS & RESULTS

- Speed Improvement: HDD reduces test cases significantly (e.g., from 680 to 86 tests).
- Simpler Outputs: Maintains syntactic structure, making minimized inputs easier to understand.
- Scalability: Handles large, complex inputs that traditional DD cannot process efficiently.
- For Mozilla's XML files, HDD reduced the number of tests by up to 85% compared to traditional methods.
- Empirical Results:
  - GCC Bugs: Orders of magnitude fewer test cases.
  - Mozilla XML Bugs: Faster minimization with simpler outputs.

```
void f()
{
    int x;  int y;
    if (x!=0) { y= x; } else { return 0; }
    while (y!=0) { y--; }
    return y;
}
```





APPLICATIONS OF HDD

- Programming Languages:
  - Debugging compiler errors using Abstract Syntax Trees.
  - Identifying minimal code snippets causing failures.
- HTML/XML Processing:
  - Simplifying deeply nested XML or HTML inputs.
  - Debugging web browsers, parsers.
- Video Codecs:
  - Identifying problematic frames in video`sequences.
  - Debugging multimedia applications.
- UI Testing:
  - Minimizing complex user interactions.
  - Reproducing GUI crashes with simplified user sessions.
- Configuration Files:
  - Simplifying JSON, YAML configurations to isolate issues.
  - Debugging software setups and deployments.

RECENT WORK BY AUTHORS

1. Hierarchical Delta Debugging (HDD): Introduced by Misherghi and Su to enhance traditional Delta Debugging by leveraging the hierarchical structure of inputs, leading to more efficient and effective debugging processes.
2. DECKARD: Developed a scalable and accurate tool for detecting code clones by analyzing tree-based representations of source code, facilitating improved code maintenance and refactoring.
3. Firewall Optimization Framework: Proposed a general framework for benchmarking firewall optimization techniques, addressing the complexity of modern firewall policies and enhancing network security management.

- Collectively, these contributions have significantly advanced methodologies in automated debugging, code analysis, and network security, influencing subsequent research and practical applications in software engineering.

PAPER COMPARISION

Delta Debugging (DD) – Zeller & Hildebrandt (2002)

- HDD extends traditional DD by addressing its inefficiencies with structured inputs like source code and XML.
- While DD treats inputs as flat lists, HDD applies DD hierarchically, significantly improving performance and minimizing failure-inducing inputs faster.
- HDD maintains syntactic validity, reducing inconclusive tests compared to DD's often syntactically invalid configurations.

Perses: Syntax-Guided Program Reduction – Sun, Li, Zhang, Gu, & Su (2018)

- Both HDD and Perses improve on DD by considering syntactic structures.
- Key Difference: HDD performs hierarchical reductions using tree structures, while Perses leverages formal syntax (grammar) to guide reductions, ensuring all intermediate reductions are syntactically valid.
- Performance Comparison: Perses outperforms HDD by producing smaller reductions (2%–45% of HDD's size) and completing reductions in 47% of HDD's time.

TECHNICAL INSIGHTS

1. Hierarchical Input Handling: The idea of treating inputs as hierarchies rather than flat lists is a simple yet powerful innovation that significantly enhances debugging efficiency.
2. Syntactic Validity: Ensuring all minimized inputs are syntactically valid reduces the number of inconclusive tests, saving time and improving results.
3. Empirical Validation: The extensive experiments using real-world bugs from GCC and Mozilla provide strong evidence of HDD's effectiveness.

LIMITATIONS

1. Complexity in Grammar Setup: While the context-free grammar approach is powerful, it may introduce complexity for users unfamiliar with formal grammars.
2. Limited to Structured Inputs: HDD is highly effective for structured data but offers little advantage for unstructured or flat input formats.
3. Tool Availability and Integration: At the time of publication, the availability of the HDD tool for broader use was limited. Better integration with existing debugging frameworks could enhance adoption.

CONCLUSION AND FUTURE WORK

Summary:

- HDD significantly improves the speed and quality of failure-inducing input minimization.
- Leverages hierarchical structures for efficient debugging.

Impact:

- Effective for debugging structured inputs like code and XML.
- Offers practical benefits in real-world scenarios.

Future Work:

- Handling Complex Dependencies: Improve the algorithm to better handle inputs with high interdependencies between data.
- Optimizing Simplification: Explore techniques like branch containment to limit ddmin to specific parent nodes for faster reduction.
- General Context-Free Grammar Framework: Create a universal CFG framework to facilitate the adoption of HDD across various domains.

Thank You!

Questions?