Model-based hierarchical delta debugging

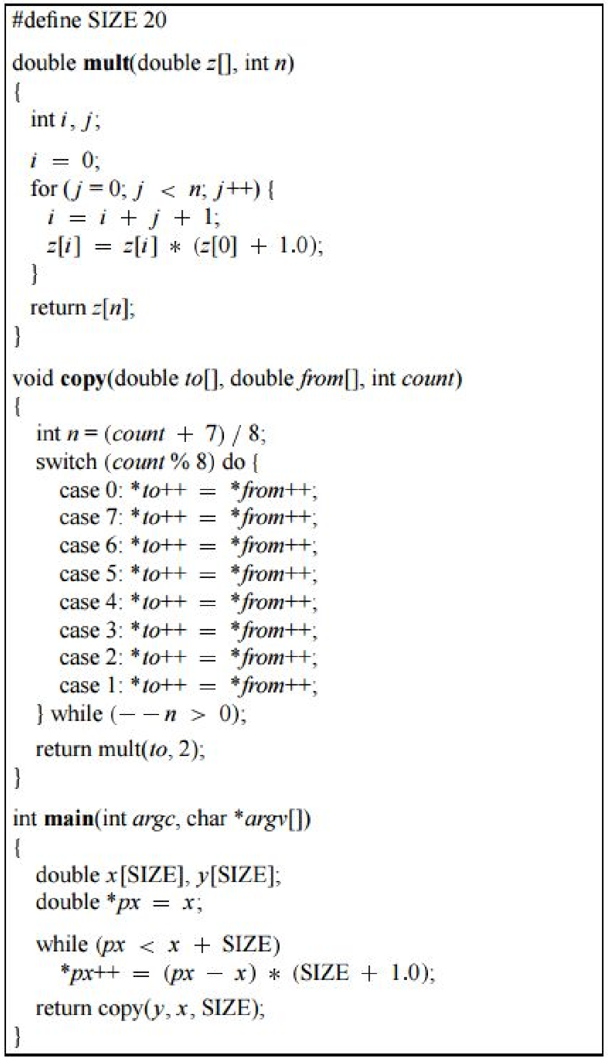
Isolating failure-inducing inputs that can be represented as a tree

Satia Herfert

Outline

1. Motivation
2. Delta debugging
3. Hierarchical Delta Debugging
4. The child substitution rule
5. Model-based HDD
6. Preliminary results
7. Outlook

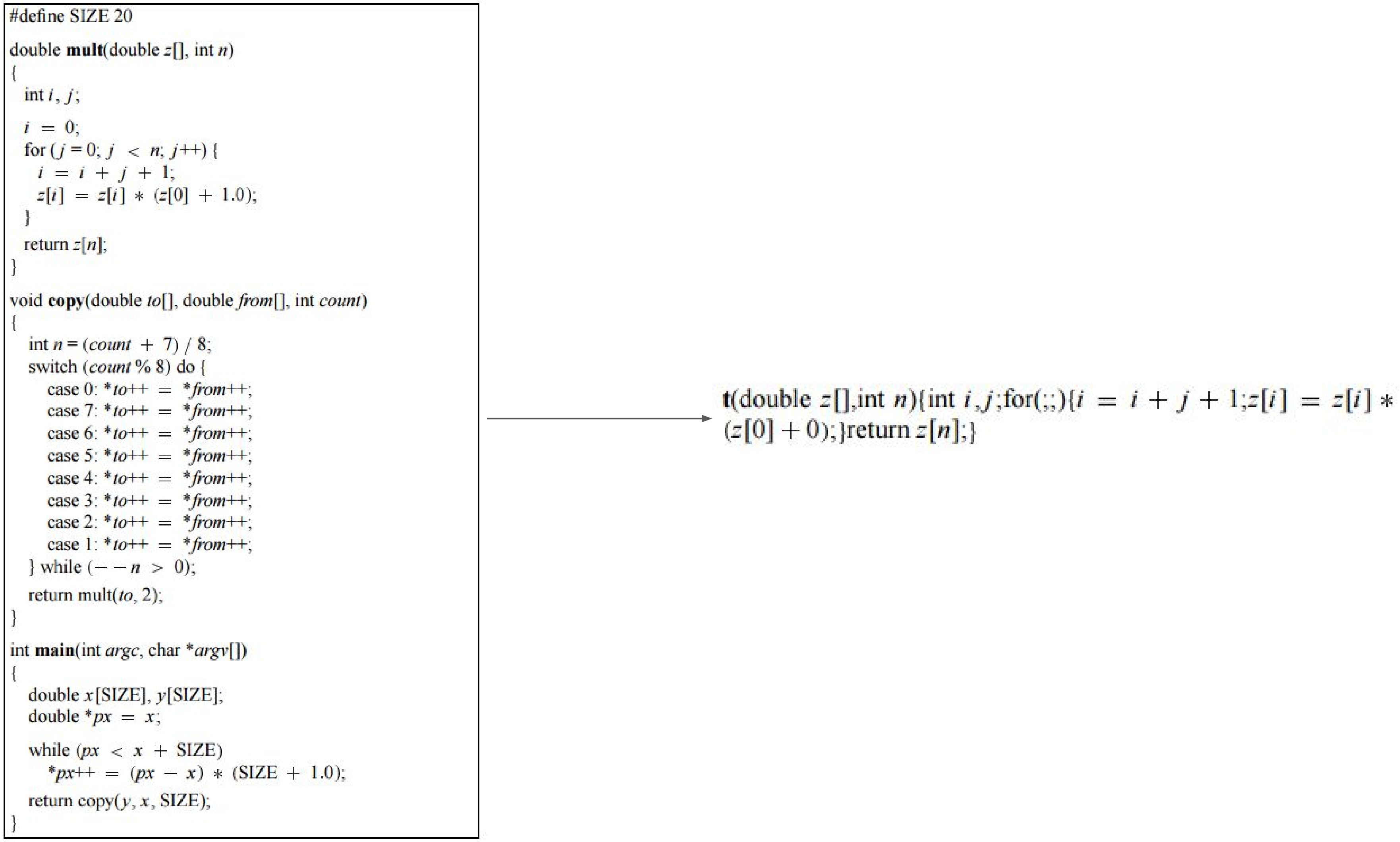
Motivation



```
#define SIZE 20

double mult(double z[], int n)
{
  int i , j ;

  i = 0;
  for (j = 0; j < n; j++) {
    i = i + j + 1;
    z[i] = z[i] * (z[0] + 1.0);
  }
  return z[n];
}

void copy(double to[], double from[], int count)
{
   int n = (count + 7) / 8;
   switch (count % 8) do {
      case 0: *to++ = *from++;
      case 7: *to++ = *from++;
      case 6: *to++ = *from++;
      case 5: *to++ = *from++;
      case 4: *to++ = *from++;
      case 3: *to++ = *from++;
      case 2: *to++ = *from++;
      case 1: *to++ = *from++;
   } while (--n > 0);
   return mult(to, 2);
}

int main(int argc, char *argv[])
{
   double x[SIZE], y[SIZE];
   double *px = x;

   while (px < x + SIZE)
      *px++ = (px - x) * (SIZE + 1.0);
   return copy(y, x, SIZE);
}
```

- **Crashes GCC 2.95.2**
- **Is this the smallest input triggering the bug?**

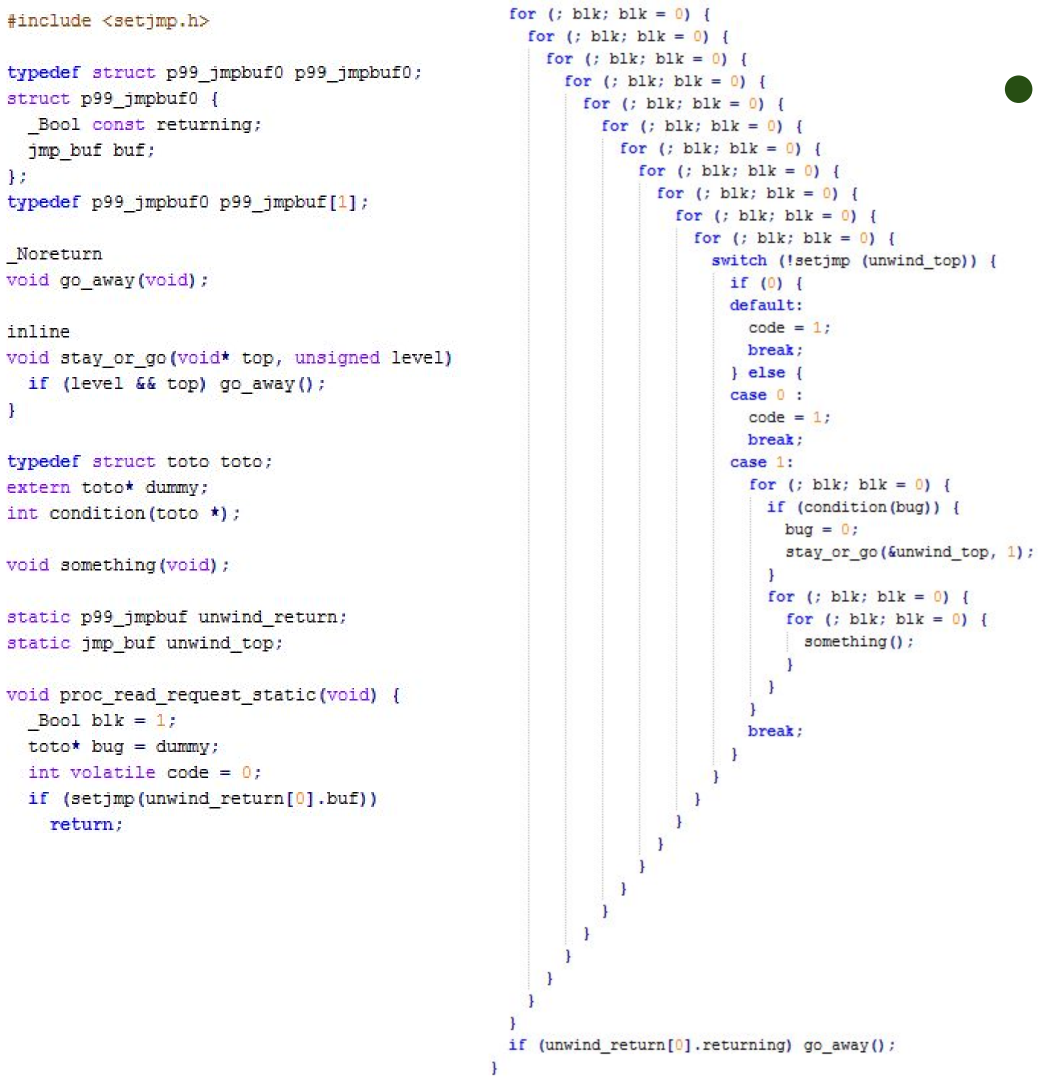
Motivation



```
#define SIZE 20

double mult(double z[], int n)
{
  int i, j;

  i = 0;
  for (j = 0; j < n; j++) {
    i = i + j + 1;
    z[i] = z[i] * (z[0] + 1.0);
  }
  return z[n];
}

void copy(double to[], double from[], int count)
{
   int n = (count + 7) / 8;
   switch (count % 8) do {
      case 0: *to++ = *from++;
      case 7: *to++ = *from++;
      case 6: *to++ = *from++;
      case 5: *to++ = *from++;
      case 4: *to++ = *from++;
      case 3: *to++ = *from++;
      case 2: *to++ = *from++;
      case 1: *to++ = *from++;
   } while (− − n > 0);
   return mult(to, 2);
}

int main(int argc, char *argv[])
{
   double x[SIZE], y[SIZE];
   double *px = x;

   while (px < x + SIZE)
      *px++ = (px − x) * (SIZE + 1.0);
   return copy(y, x, SIZE);
}
```

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

Motivation



```c
#include <setjmp.h>

typedef struct p99_jmpbuf0 p99_jmpbuf0;
struct p99_jmpbuf0 {
  _Bool const returning;
  jmp_buf buf;
};
typedef p99_jmpbuf0 p99_jmpbuf[1];

_Noreturn
void go_away(void);

inline
void stay_or_go(void* top, unsigned level)
  if (level && top) go_away();
}

typedef struct toto toto;
extern toto* dummy;
int condition(toto *);

void something(void);

static p99_jmpbuf unwind_return;
static jmp_buf unwind_top;

void proc_read_request_static(void) {
  _Bool blk = 1;
  toto* bug = dummy;
  int volatile code = 0;
  if (setjmp(unwind_return[0].buf))
    return;
  for (; blk; blk = 0) {
    for (; blk; blk = 0) {
      for (; blk; blk = 0) {
        for (; blk; blk = 0) {
          for (; blk; blk = 0) {
            for (; blk; blk = 0) {
              for (; blk; blk = 0) {
                for (; blk; blk = 0) {
                  for (; blk; blk = 0) {
                    for (; blk; blk = 0) {
                      for (; blk; blk = 0) {
                        switch (!setjmp (unwind_top)) {
                          if (0) {
                          default:
                            code = 1;
                            break;
                          } else {
                          case 0 :
                            code = 1;
                            break;
                          case 1:
                            for (; blk; blk = 0) {
                              if (condition(bug)) {
                                bug = 0;
                                stay_or_go(&unwind_top, 1);
                              }
                              for (; blk; blk = 0) {
                                for (; blk; blk = 0) {
                                  something();
                                }
                              }
                            }
                            break;
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
  if (unwind_return[0].returning) go_away();
}
```

- **Actual bug report**
  - https://gcc.gnu.org/bugzilla/show_bug.cgi?id=65395
  - GCC 4.9 crashes with a segmentation fault
  - Fixed 03.08.2016

Outline

1. **Motivation**
2. **Delta debugging** ←
3. **Hierarchical Delta Debugging**
4. **The child substitution rule**
5. **Model-based HDD**
6. **Preliminary results**
7. **Outlook**

Delta Debugging
Properties

- **Isolate failure-inducing inputs**
- **Fully automated**
  - Input and oracle required
- **Language independent**
- **No semantic knowledge**

Delta Debugging

Algorithm

Input	🟢	🟢	🟢	🟢	⚡
Test #1	🟢	🟢	🔴	🔴	
Test #2	🔴	🔴	🟢	🟢	
Test #3	🟢	🔴	🔴	🔴	
Test #4	🔴	🟢	🔴	🔴	
Test #5	🔴	🔴	🟢	🔴	
Test #6	🔴	🔴	🔴	🟢	
Test #7	🔴	🟢	🟢	🟢	
Test #8	🟢	🔴	🟢	🟢	⚡
Test #9	🟢	🔴	🔴	🔴	
Test #10	🔴	🔴	🟢	🔴	
Test #11	🔴	🔴	🔴	🟢	
Test #12	🔴	🔴	🟢	🟢	
Test #13	🟢	🔴	🔴	🟢	
Test #14	🟢	🔴	🟢	🔴	⚡
Test #15	🟢	🔴	🔴	🔴	
Test #16	🔴	🔴	🟢	🔴	
Result	🟢	🔴	🟢	🔴	⚡

Delta Debugging
1-Minimality

“... if removing any single change would cause the failure to disappear.”

- This does not say anything about removing 2 or more changes

Delta Debugging
What are tokens?

- **Lines**
- **Characters**
- **Bytes**
- **...**

Delta Debugging

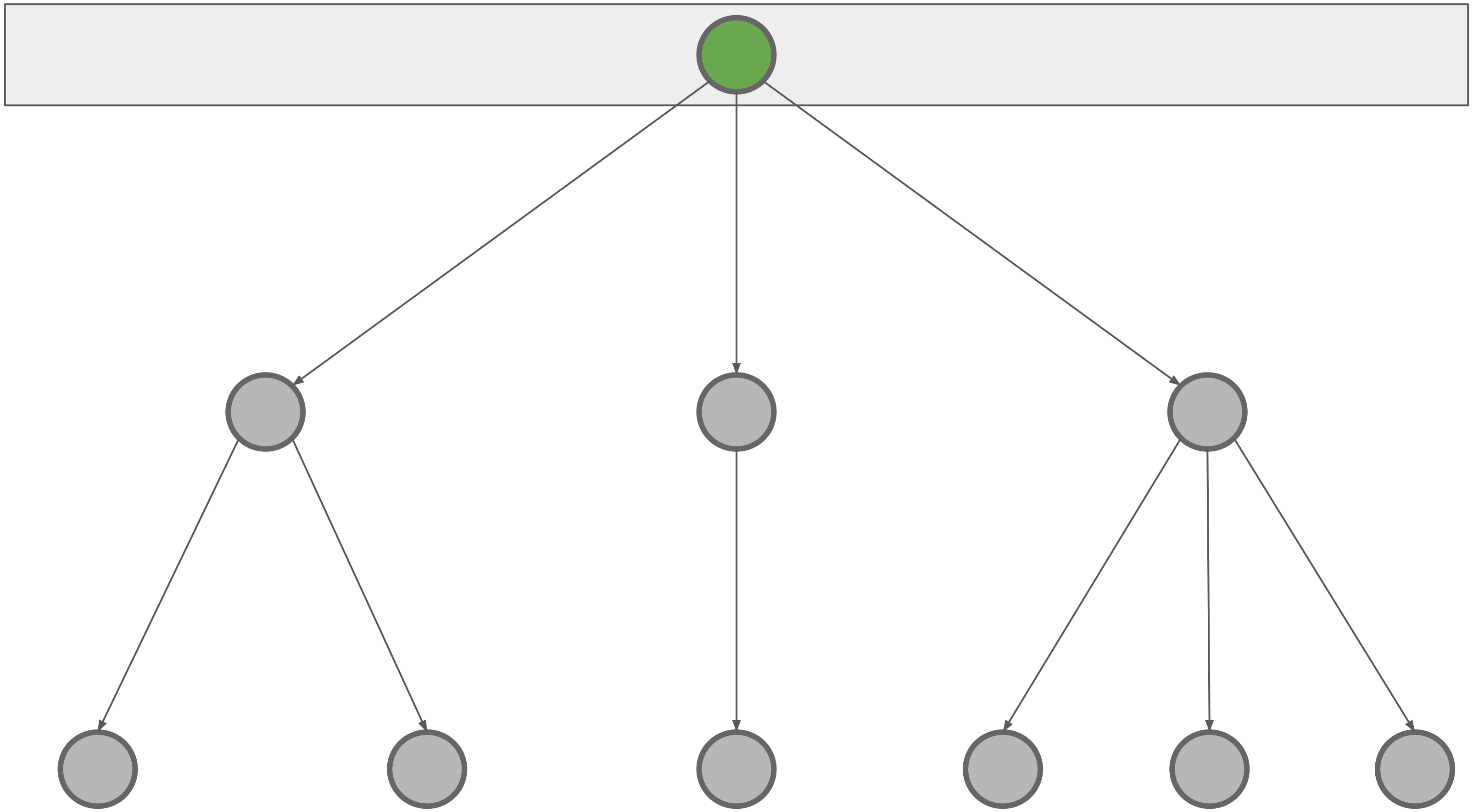
Shortcomings

- **Produces many invalid test cases**
- **Disregards structure of the document**

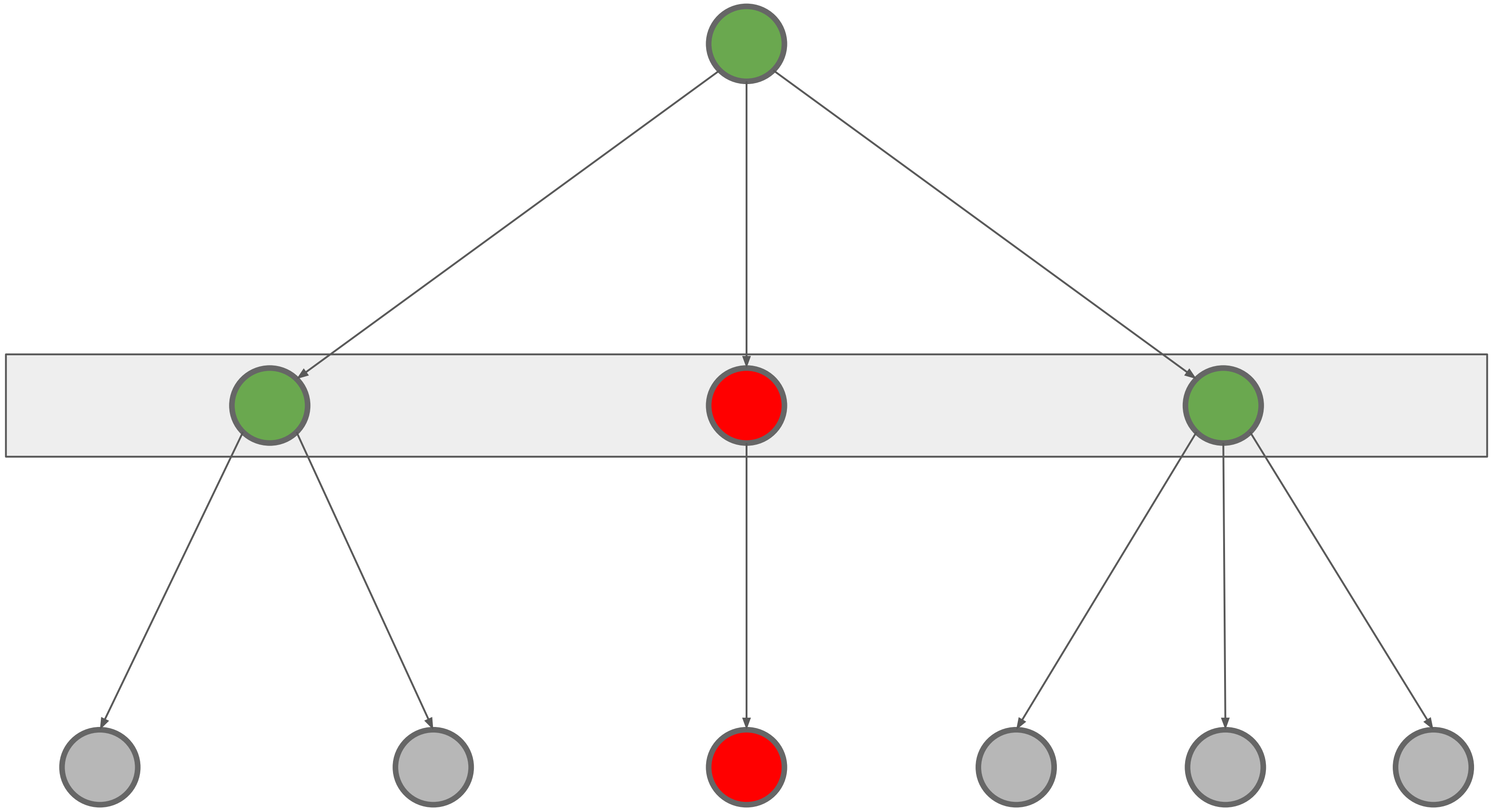
Outline

1. **Motivation**
2. **Delta debugging**
3. **Hierarchical Delta Debugging** ←
4. **The child substitution rule**
5. **Model-based HDD**
6. **Preliminary results**
7. **Outlook**

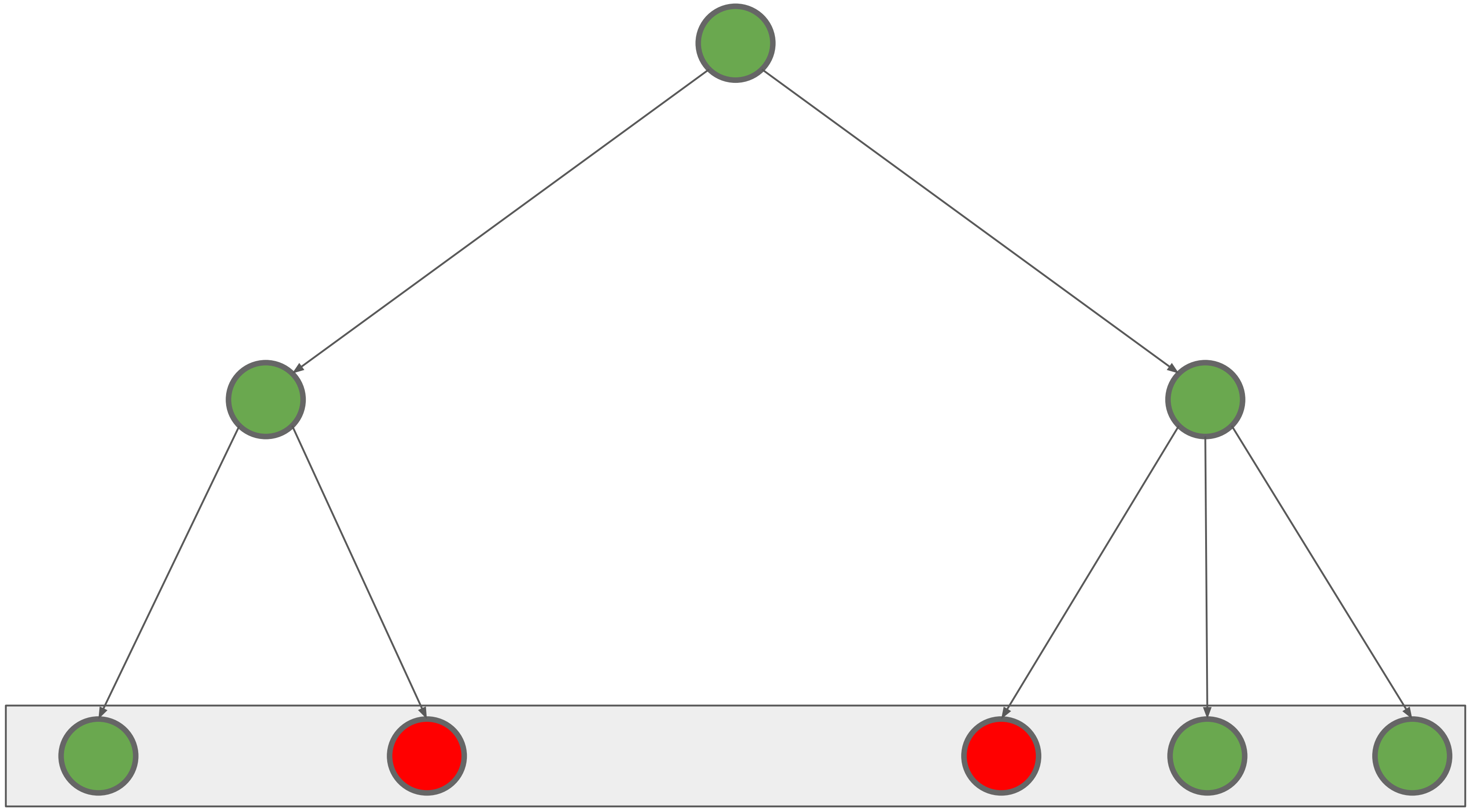
HDD
Algorithm



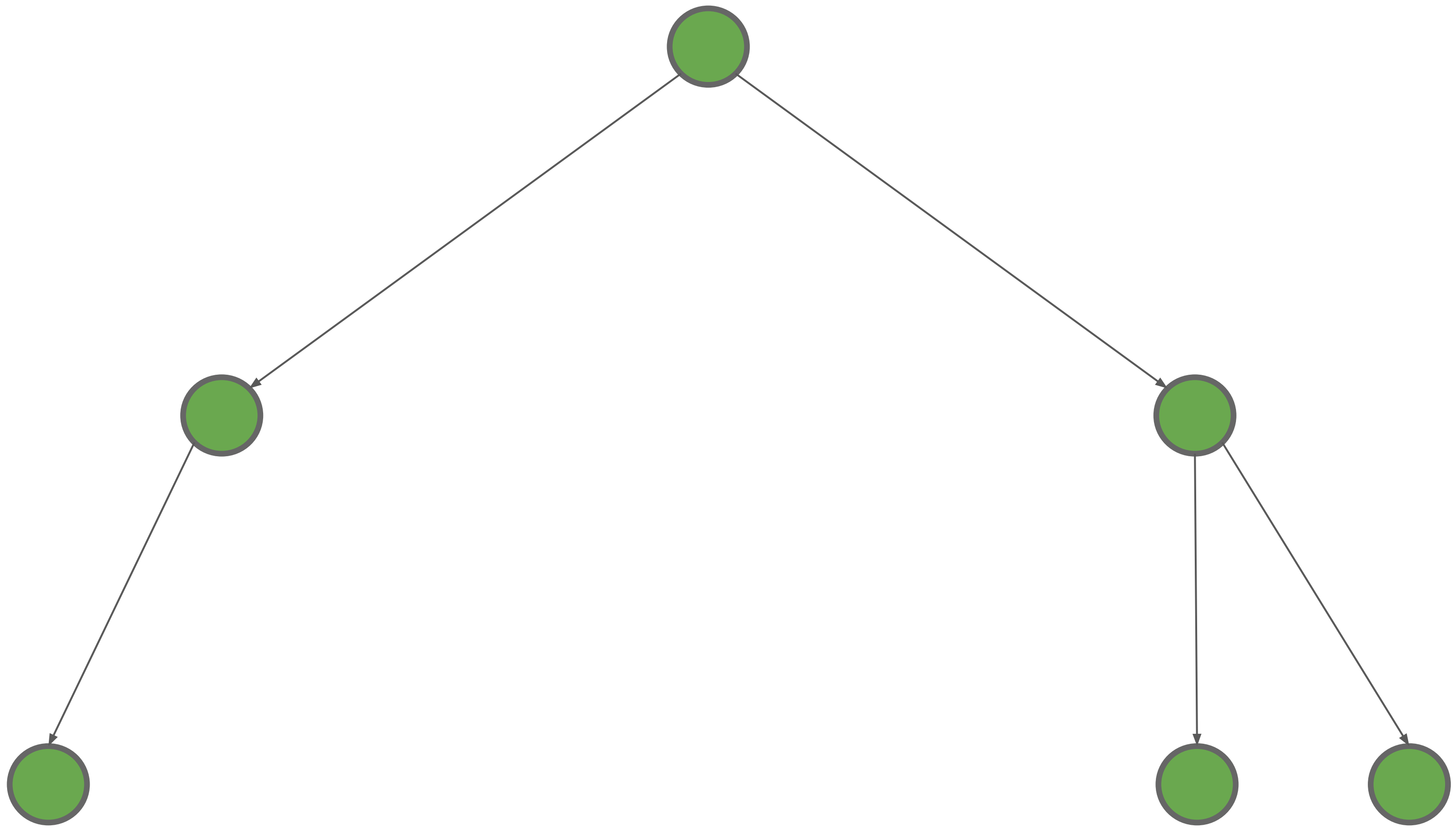
HDD
Algorithm



HDD
Algorithm



HDD
Algorithm

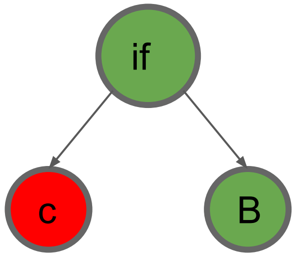


HDD
Properties

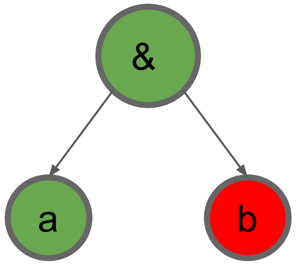
- **Needs fewer tests**
- **Produces smaller results**
- **Does not ensure 1-minimality**
- **HDD***
    - Repeat HDD until no more changes

HDD
Shortcomings

```
if(condition) {
        bug;
}
```



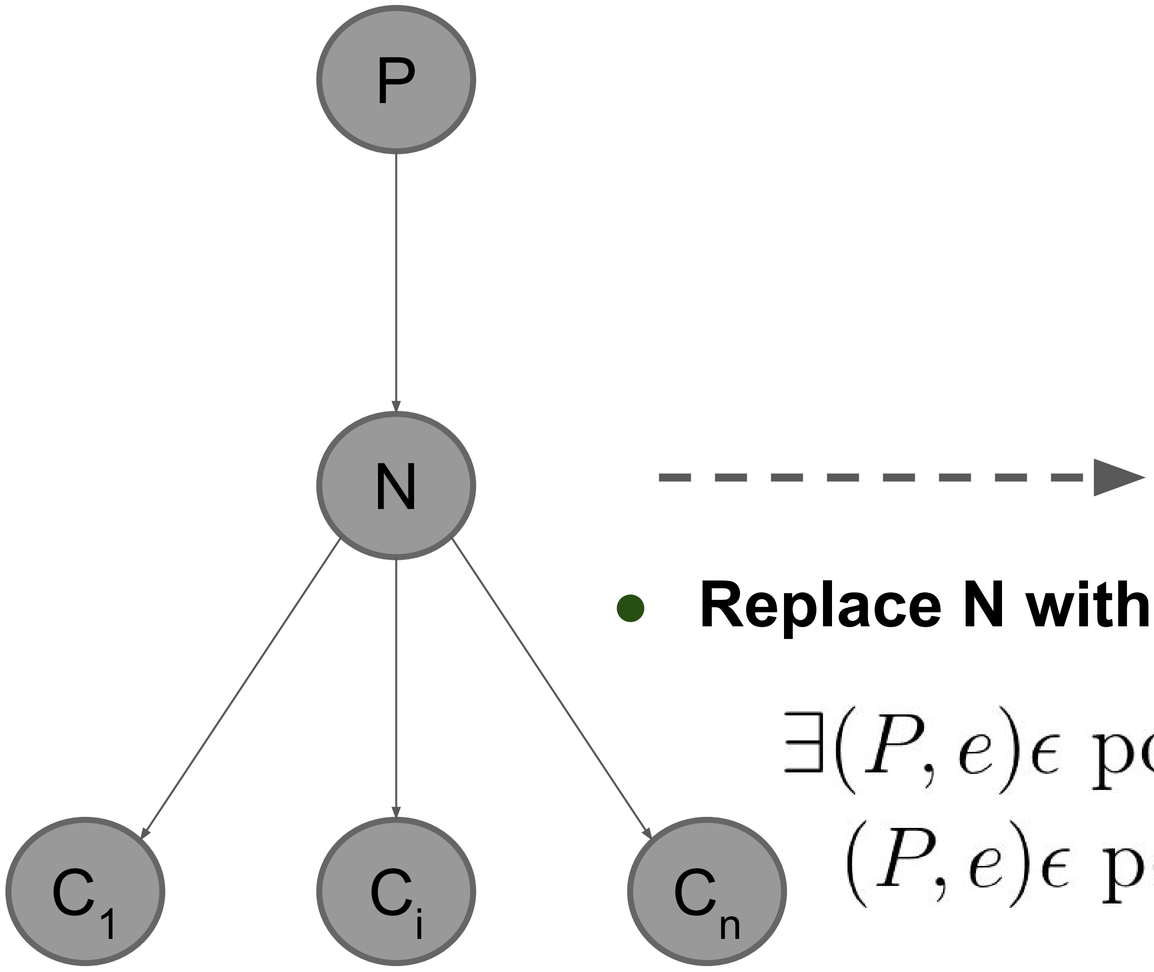
```
a && b
```



Outline

1. **Motivation**
2. **Delta debugging**
3. **Hierarchical Delta Debugging**
4. **The child substitution rule** ←
5. **Model-based HDD**
6. **Preliminary results**
7. **Outlook**

Child substitution rule



- **Replace N with its child C iff:**

$$\exists (P, e) \epsilon \text{ possibleParents}(N)$$
$$(P, e) \epsilon \text{ possibleParents}(C)$$

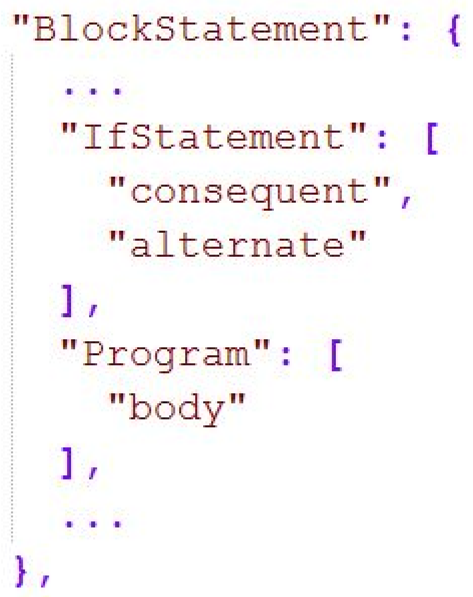
Child substitution rule

Model inference

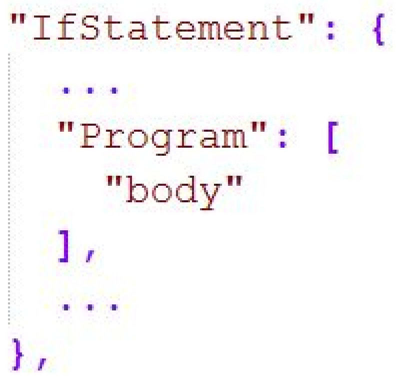
- Go through a large code base of the target language
- Collect the (P,e) possible parents of all nodes with a certain label.
- Calculate all concrete substitution rules.

Child substitution rule

Model inference



```
"BlockStatement": {
  ...
  "IfStatement": [
    "consequent",
    "alternate"
  ],
  "Program": [
    "body"
  ],
  ...
},
```

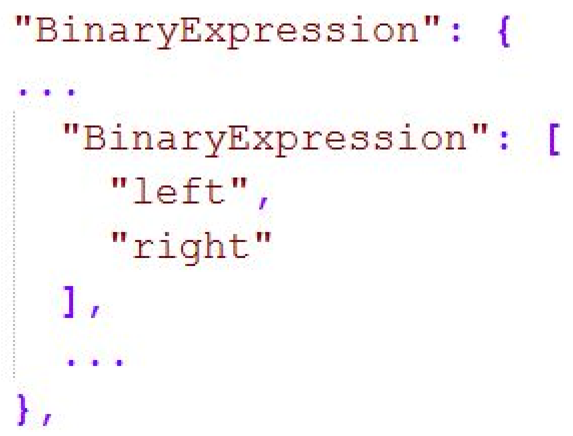


```
"IfStatement": {
  ...
  "Program": [
    "body"
  ],
  ...
},
```

- **Replace *IfStatement* with its child *consequent***
- **Replace *IfStatement* with its child *alternate***

Child substitution rule

Model inference



```
"BinaryExpression": {
...
  "BinaryExpression": [
    "left",
    "right"
  ],
  ...
},
```

- **Replace *BinaryExpression* with its child *left***
- **Replace *BinaryExpression* with its child *right***

Child substitution rule

Convergence of rules

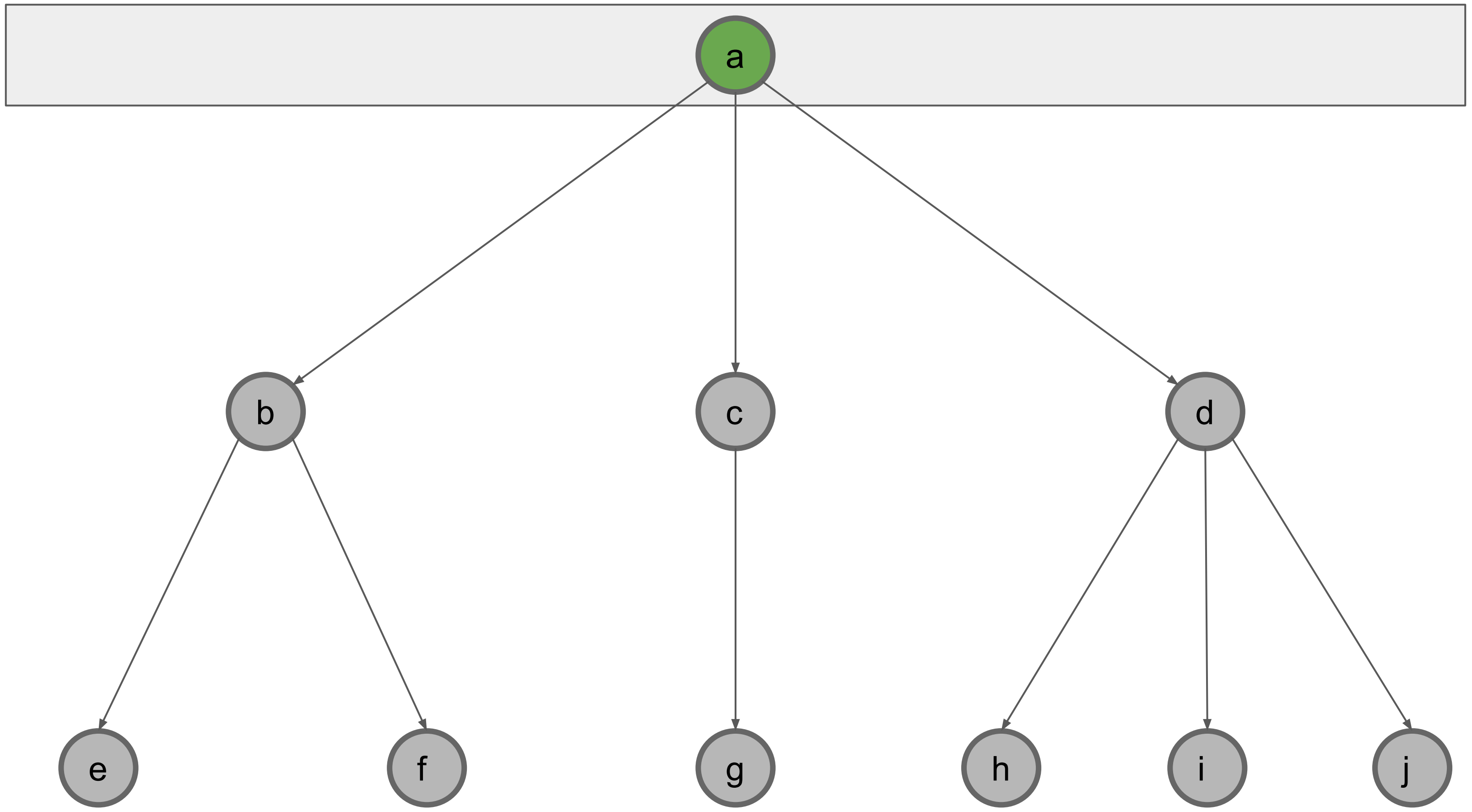
- **TODO**

Outline

1. Motivation
2. Delta debugging
3. Hierarchical Delta Debugging
4. The child substitution rule
5. Model-based HDD
6. Preliminary results
7. Outlook

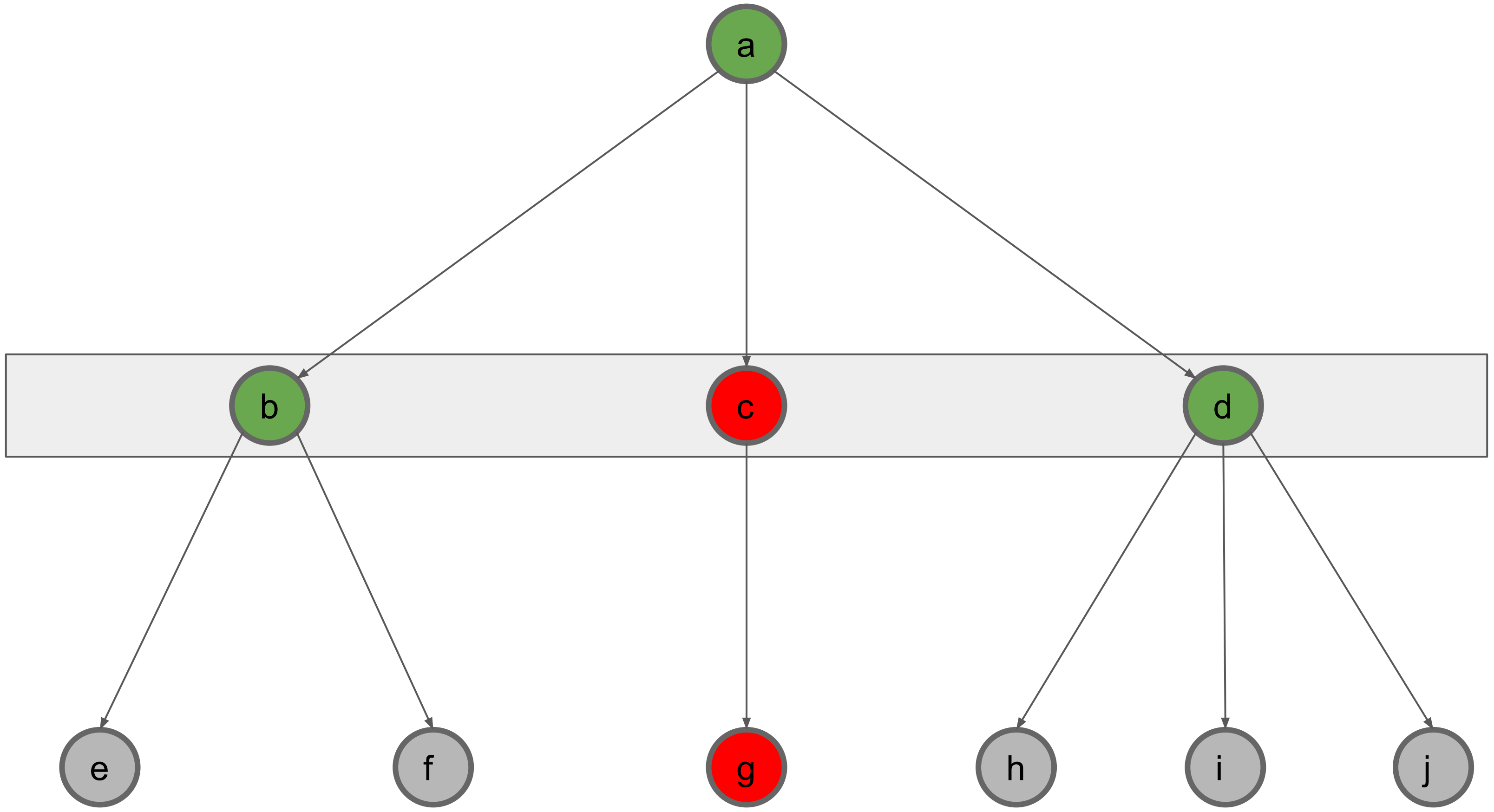
Model-based HDD

Algorithm



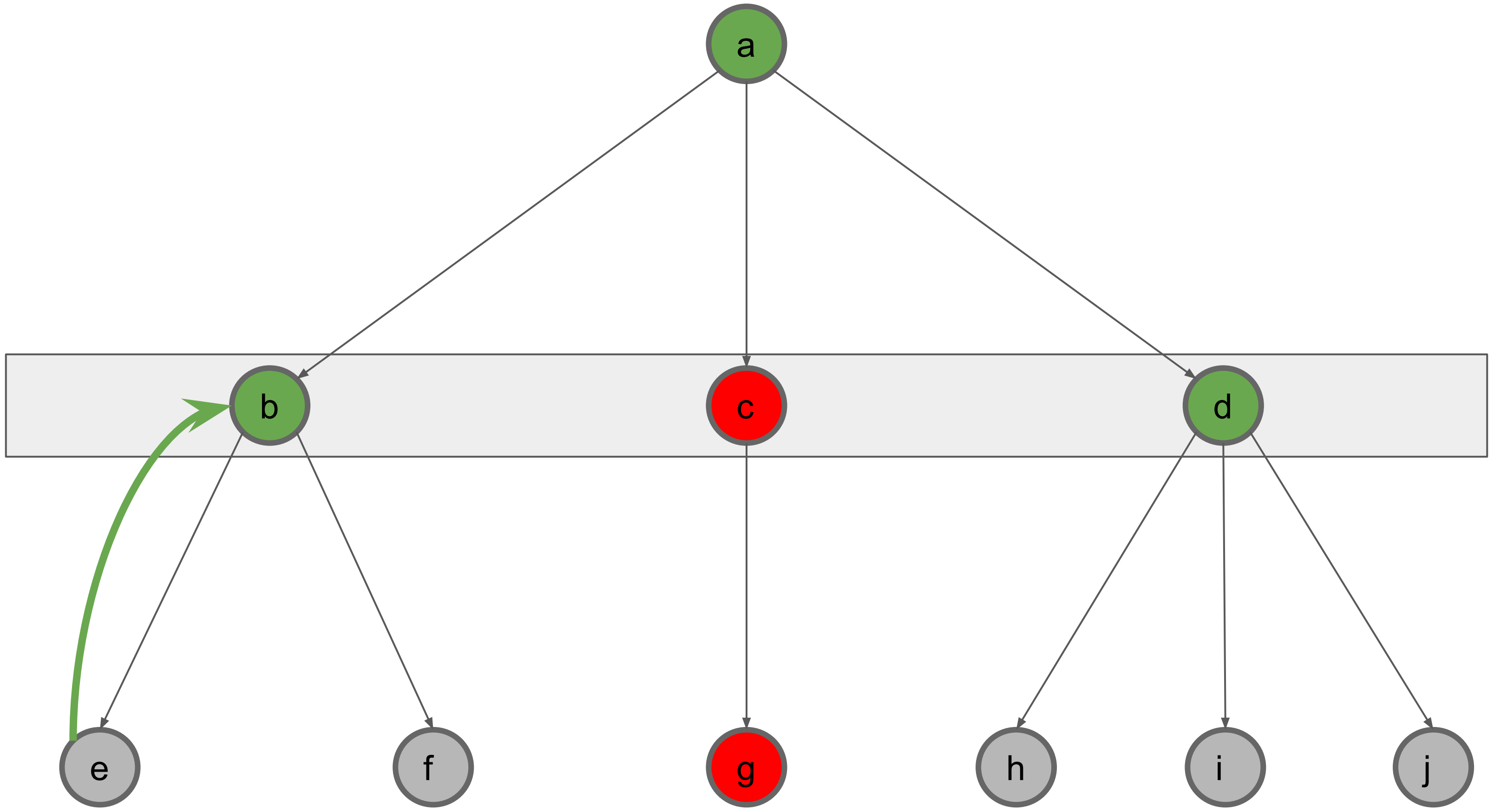
Model-based HDD

Algorithm



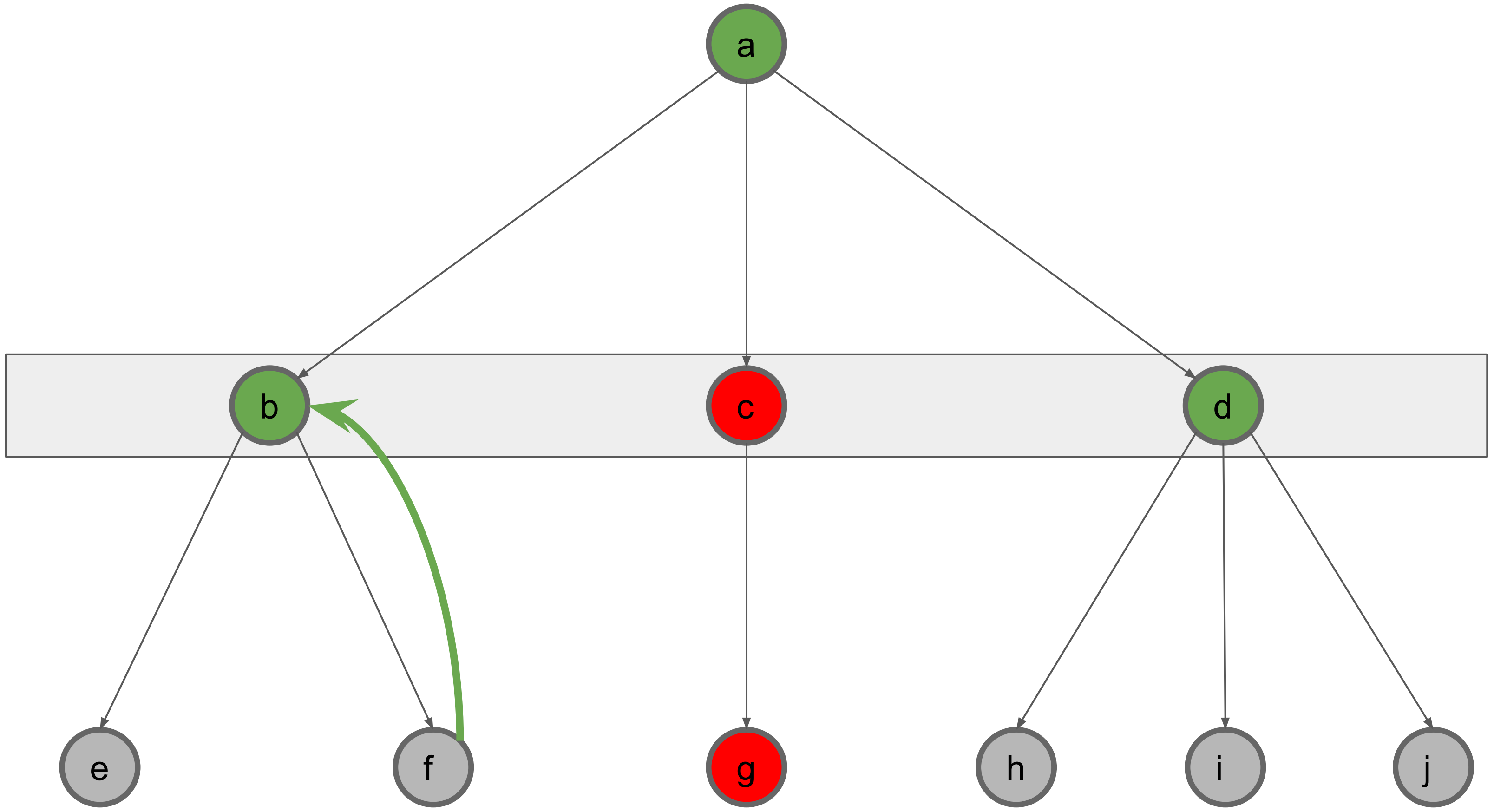
Model-based HDD

Algorithm



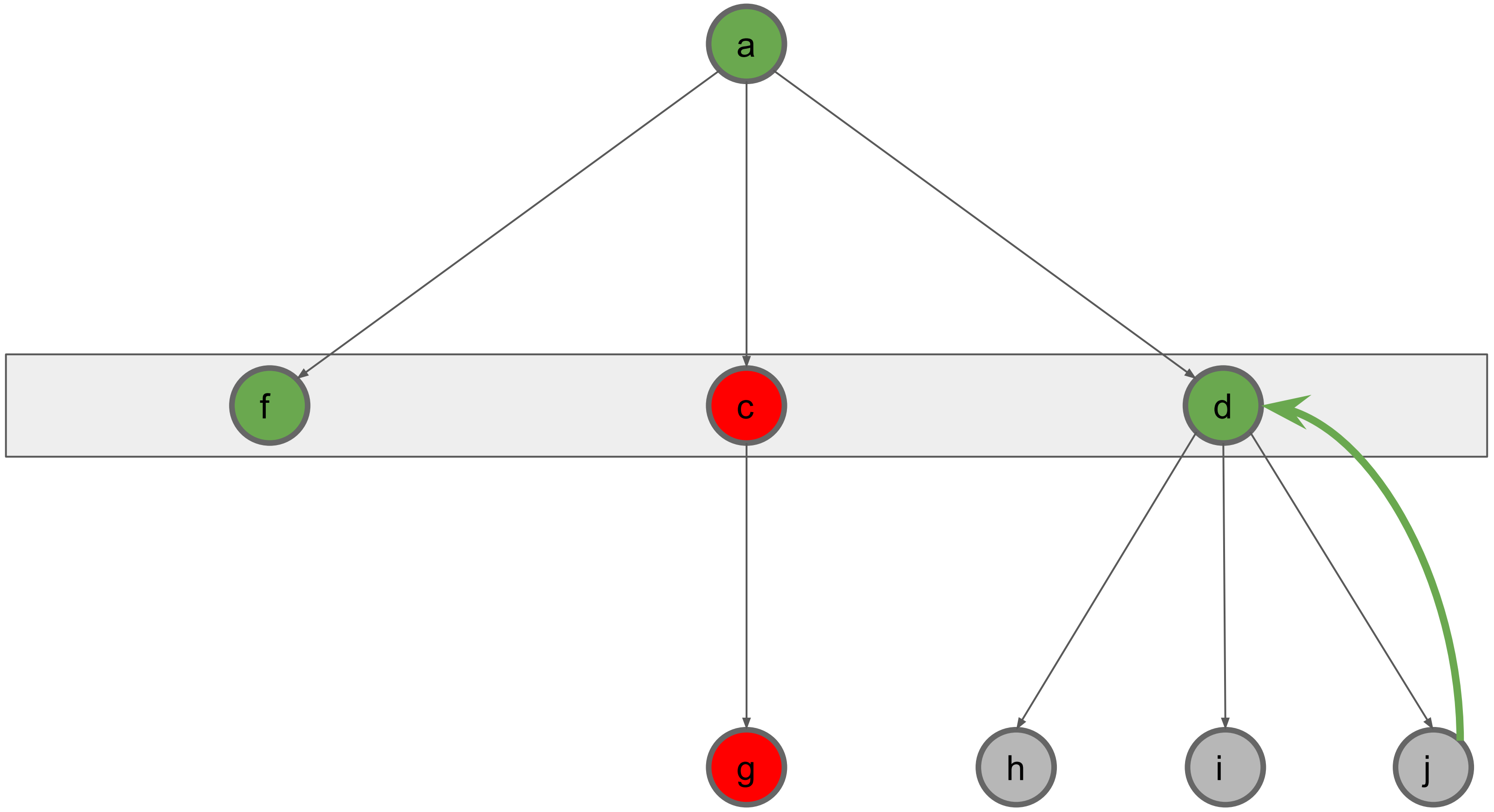
Model-based HDD

Algorithm



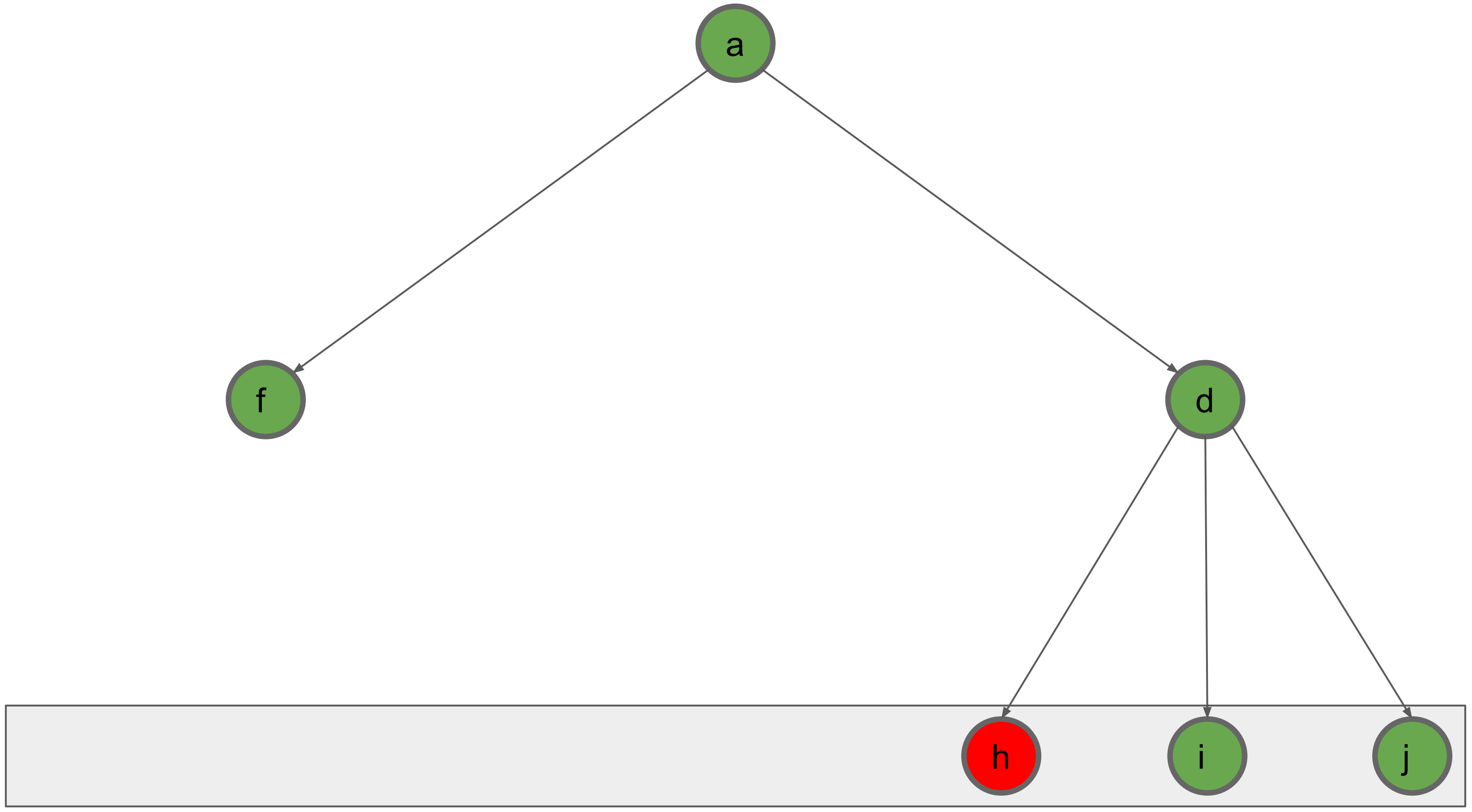
Model-based HDD

Algorithm



Model-based HDD

Algorithm



Outline

1. **Motivation**
2. **Delta debugging**
3. **Hierarchical Delta Debugging**
4. **The child substitution rule**
5. **Model-based HDD**
6. **Preliminary results** ←
7. **Outlook**

Preliminary results

Comparing against HDD

- **Testing with 44 JavaScript files**
- **Exposing an inconsistency across browsers**
- **TODO**
  - Statistics
  - Image

Outline

1. **Motivation**
2. **Delta debugging**
3. **Hierarchical Delta Debugging**
4. **The child substitution rule**
5. **Model-based HDD**
6. **Preliminary results**
7. **Outlook** ←

Outlook

- **Cross-check the results with a different language**
- **Infer more rules than just the child substitution rule**
- **Integrate the transformations less naive into the algorithm**