Regular Property Guided Dynamic Symbolic Execution

Zhenbang Chen
(zbchen@nudt.edu.cn)
joint work with Yufeng Zhang, Ji Wang, Wei Dong and Zhiming Liu

College of Computer, National University of Defense Technology, China
Centre for Software Engineering, Birmingham City University, UK

2015.05.22
Dynamic Symbolic Execution (DSE)

• Explore path spaces systematically
  • Test-case generation, bug-finding, bounded verification, …

• Path explosion problem
DSE needs guiding
DSE needs guiding
Existing Work of Guiding Symbolic Execution

• Improving coverage
  • KLEE[OSDI’08], CREST[ASE’08], SGS[OOPSLA’13], CGS[FSE’14], …

• Reach program points
  • PEX[DSN’09], ESD[EuroSys’11], SDSE [SAS’11], BitBlaze[ISSTA’11], …

• Exploring the difference between programs
  • DiSE[PLDI’11], ZESTI [ICSE’12], KATCH[FSE’13], …
How about a Regular Property?
How about a Regular Property?

A bug property: a file is read after closed
How about a Regular Property?

A bug property: a file is read after closed

Program \( \cap \) Regular Property P (FSM) \( \neq \emptyset \) ?
How about a Regular Property?

A bug property: a file is read after closed

Program $\cap$ Regular Property $P$ (FSM) $\neq \emptyset$ ?

How to guide DSE to find a program path satisfying $P$ as soon as possible?
Observation and Insight

• Many irrelevant paths exist
• Even for relevant paths, only the ones with specific sequences can satisfy the regular property
Observation and Insight

• Many irrelevant paths exist
• Even for relevant paths, only the ones with specific sequences can satisfy the regular property

Evaluate the possibility of a branch to generate the paths satisfying the property
Key Idea

Evaluate a branch based on its history and future behaviors
Key Idea

Evaluate a branch based on its history and future behaviors
Key Idea

Evaluate a branch based on its history and future behaviors
Evaluate a branch based on its history and future behaviors
Key Idea

history \cap \text{future} \neq \emptyset

Evaluate a branch based on its history and future behaviors
Key Idea

Evaluate a branch based on its history and future behaviors
Key Idea

Evaluate a branch based on its history and future behaviors

**Preset:** the state that can be reached from the beginning to the branch location

Dynamic analysis

\[ \text{history} \cap \text{future} \neq \emptyset \]
Key Idea

Evaluate a branch based on its history and future behaviors.

Preset: the state that can be reached from the beginning to the branch location.

Dynamic analysis:

Postset: the states from which it can reach a final state after executing the rest of the program after the branch location.

Static analysis:

\[ \text{history} \cap \text{future} \neq \emptyset \]
Sneak Preview of Results

• For finding the first accepted path
  • >1880X speedup on iterations
  • >258X time speedup on the programs whose paths space is bigger than 100
• For 3 out of the 13 real world programs
  • Guided method succeeds in 1 hour
  • Pure method fails in 24 hours
Procedure

```
int foo(int m, n, tag) {
    InputStreamReader w = new ...
    int result = 0, k = 0, i = -1;
    while (k++ < m)
        i = w.read();
        if (i == -1)
            break;
        result += i;
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n)
        i = w.read();
        if (i == -1)
            break;
        result -= i;
    return result;
}
```
An Example

```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n){
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
```

Reader Property

A reader is read after closed
An Example

```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n){
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
```

A reader is read after closed

Reader Property
An Example

```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n) {
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
```

Reader Property

A reader is read after closed
An Example

```
int foo(int m, n, tag) {
  InputStreamReader w = new ...
  int result = 0, k = 0, i = -1;
  while (k++ < m)
    {
      i = w.read();
      if (i == -1) break;
      result += i;
    }
  if (tag == 0) w.close();
  k = 0;
  while (k++ < n) {
    i = w.read();
    if (i == -1) break;
    result -= i;
  }
  return result;
}
```

Reader Property

A reader is read after closed
An Example

```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1)
            break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n){
        i = w.read();
        if (i == -1)
            break;
        result -= i;
    }
    return result;
}
```

Reader Property

A reader is read after closed
Pure DSE
```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n){
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
```

(m=1, n=1, tag=1)

DFS

m > 0
m ≤ 1
tag != 0
n > 0
n ≤ 1
m > 0 ∧ m ≤ 1 ∧ tag != 0 ∧ n > 0 ∧ n ≤ 1
Pure DSE-1st Iteration

```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m) {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n) {
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
```

DFS

\[(m=1, \ n=1, \ tag=1)\]

\[m > 0\]

\[m \leq 1\]

\[tag \neq 0\]

\[n > 0\]

\[n \leq 1\]

\[n > 1\]

\[m > 0 \land m \leq 1 \land tag \neq 0 \land n > 0 \land n \leq 1\]

\[m > 0 \land m \leq 1 \land tag \neq 0 \land n > 0 \land n > 1\]

\[(m=1, \ n=2, \ tag=1)\]
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n){
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n){
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1)
            break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n)
    {
        i = w.read();
        if (i == -1)
            break;
        result -= i;
    }
    return result;
}
Guided DSE
Guided DSE Procedure

(1) Static analysis

(2) DSE

Running DSE

&

Finished

Input generation

Next branch selection

Report results
Postset Calculation
```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0)
        w.close();
    k = 0;
    while (k++ < n)
    {
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
```

**Reader Property**

0
init →
1
read →
2
close →
3
read, close

**Postset Calculation**

Backward data flow analysis [Clara, ICSE’10]

$O(|E| \times |D|^3)$
```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
}
```

Reader Property

```
0
init

1
read

close

2
close

3
read
read, close
```

Backward IFDS data flow analysis

$O(|E| \times |D|^3)$
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
}
```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
}
```

Reader Property

0
init

1  read

2  close

3  read, close

Backward IFDS data flow analysis

$O(|E| \times |D|^3)$
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
}
```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
}
```

**Reader Property**

```
0 -> init
1 -> read
2 -> close
3 -> read, close
```

**Backward IFDS data flow analysis**

\[ O(|E| \times |D|^3) \]
Guided DSE Procedure

```
int foo(int m, n, tag) {
    InputStreamReader w = new ...
    int result = 0, k = 0, i = -1;
    while (k++ < m) {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n) {
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
```
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n)
    {
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}  

Preset-1st Iteration

(m=1, n=1, tag=1)
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m) {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n) {
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
int foo(int m, n, tag) {
    InputStreamReader w = new InputStreamReader(...);
    int result = 0, k = 0, i = -1;
    while (k++ < m) {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n) {
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m) {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n) {
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
int foo(int m, n, tag) {
    InputStreamReader w = new ...;
    int result = 0, k = 0, i = -1;
    while (k++ < m)
    {
        i = w.read();
        if (i == -1) break;
        result += i;
    }
    if (tag == 0) w.close();
    k = 0;
    while (k++ < n){
        i = w.read();
        if (i == -1) break;
        result -= i;
    }
    return result;
}
Guided DSE-1st Iteration

(int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
})

Guided DSE-1st Iteration

(m=1, n=1, tag=1)

Guided DSE-1st Iteration

(int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
})
Guided DSE-1st Iteration

(m=1, n=1, tag=1)

```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
}
```
Guided DSE-1st Iteration

\[(m=1, n=1, tag=1)\]

```
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
}
```
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
}
Guided DSE-1st Iteration

int foo(int m, n, tag) {
  InputStreamReader w = new ...; //{0}
  int result = 0, k = 0, i = -1; //{1,2,3}
  while (k++ < m) //{1,2,3}
  {
    i = w.read(); //{1,2,3}
    if (i == -1) break; //{1,2,3}
    result += i; //{1,2,3}
  }
  if (tag == 0) //{1,2,3}
    w.close(); //{1,2,3}
  k = 0; //{2,3}
  while (k++ < n) //{2,3}
  {
    i = w.read(); //{2,3}
    if (i == -1) break; //{2,3}
    result -= i; //{2,3}
  }
  return result; //{3}
}
Guided DSE-1st Iteration

```
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //{0}
    int result = 0, k = 0, i = -1; //{1,2,3}
    while (k++ < m) //{1,2,3}
    {
        i = w.read(); //{1,2,3}
        if (i == -1) break; //{1,2,3}
        result += i; //{1,2,3}
    }
    if (tag == 0) //{1,2,3}
        w.close(); //{1,2,3}
    k = 0; //{2,3}
    while (k++ < n) //{2,3}
    {
        i = w.read(); //{2,3}
        if (i == -1) break; //{2,3}
        result -= i; //{2,3}
    }
    return result; //{3}
}
```

(m=1, n=1, tag=1)
int foo(int m, n, tag) {  
  InputStreamReader w = new ...; //{0}  
  int result = 0, k = 0, i = -1; //{1,2,3}  
  while (k++ < m) //{1,2,3}  
  {  
    i = w.read(); //{1,2,3}  
    if (i == -1) break; //{1,2,3}  
    result += i; //{1,2,3}  
  }  
  if (tag == 0) //{1,2,3}  
    w.close(); //{1,2,3}  
  k = 0; //{2,3}  
  while (k++ < n) //{2,3}  
  {  
    i = w.read(); //{2,3}  
    if (i == -1) break; //{2,3}  
    result -= i; //{2,3}  
  }  
  return result; //{3}  
}
int foo(int m, n, tag) {
    InputStreamReader w = new ...; // {0}
    int result = 0, k = 0, i = -1; // {1,2,3}
    while (k++ < m) // {1,2,3}
    {
        i = w.read(); // {1,2,3}
        if (i == -1) break; // {1,2,3}
        result += i; // {1,2,3}
    }
    if (tag == 0) // {1,2,3}
        w.close(); // {1,2,3}
    k = 0; // {2,3}
    while (k++ < n) // {2,3}
    {
        i = w.read(); // {2,3}
        if (i == -1) break; // {2,3}
        result -= i; // {2,3}
    }
    return result; // {3}
}
Guided DSE

\[(m=1, n=1, \text{tag}=0)\]

```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //\{0\}
    int result = 0, k = 0, i = -1; //\{1,2,3\}
    while (k++ < m) //\{1,2,3\}
    {
        i = w.read(); //\{1,2,3\}
        if (i == -1) break; //\{1,2,3\}
        result += i; //\{1,2,3\}
    }
    if (tag == 0) //\{1,2,3\}
        w.close(); //\{1,2,3\}
    k = 0; //\{2,3\}
    while (k++ < n) //\{2,3\}
    {
        i = w.read(); //\{2,3\}
        if (i == -1) break; //\{2,3\}
        result -= i; //\{2,3\}
    }
    return result; //\{3\}
}
```
Guided DSE

\[(m=1, n=1, \text{tag}=0)\]

Only 2 iterations

```java
int foo(int m, n, tag) {
    InputStreamReader w = new ...; //\{0\}
    int result = 0, k = 0, i = -1; //\{1,2,3\}
    while (k++ < m) //\{1,2,3\}
    {
        i = w.read(); //\{1,2,3\}
        if (i == -1) break; //\{1,2,3\}
        result += i; //\{1,2,3\}
    }
    if (tag == 0) //\{1,2,3\}
        w.close(); //\{1,2,3\}
    k = 0; //\{2,3\}
    while (k++ < n) //\{2,3\}
    {
        i = w.read(); //\{2,3\}
        if (i == -1) break; //\{2,3\}
        result -= i; //\{2,3\}
    }
    return result; //\{3\}
}
```
Implementation & Experiment Setup

• Implement based on JPF-JDart and WALA
• 13 real world open source Java programs
  • 225K LOC in total
• Properties
  • Typestate bug && User defined
• Analyze each program/property in 24 hours
Evaluate Guiding Further

- Relevant path
- Transition times
- Shortest distance to the final state
Relevant path distribution

Number of relevant paths vs Time(s)

Guided

Pure

Time(s)
Guided DSE explore more relevant paths than the pure DSE, and earlier.
Hence, from the global view, the number of the relevant paths increases after 2000 seconds (10 minutes). On the other hand, for the pure method, the beginning, and the number decreases after around 600 second and refine methods explore more relevant paths at the be-

vant paths more earlier, which may result in finding accepted of the relevant path, because the property is a reachability case of the pure method. For the state transition metric, we record the state tran-

ric reflects one of the abilities to drive a program to-

The times of the state transitions in a path. This met-

Value(i) = \sum_{c \in \text{Combinations}} ST_g^c(i) - \sum_{c \in \text{Combinations}} ST_p^c(i)

In Figure 5, the X-axis is the path number, and Y-axis is the number of the relevant paths that begin to be explored during analyses.

In addition to the percentage, we also want to explore rele-

As shown in Table 2, our guided DSE successfully finds a path satisfying the property for each combination. In one case, the number in the brackets is when the first path satisfying the property is found; the number in the brackets is the time used for static analysis. The four columns in the table indicate the per-

The "big" column show the numbers of the total relevant paths, the "rel" column show the numbers of the relevant paths that begin to be explored during each analysis, re-

Figure 4 shows the relevant path distribution during one analysis task by allocating 10GB memory to the JVM, 256GB memory and four 2.13GHz XEON CPUs. We run the "refine", the three kinds of analyses are DSE with-

However, the metrics of relevant paths are still not enough to guiding (<i>g</i>) or refine (<i>r</i>) method, respectively. We select the first 50000 paths for observation. In the same way, we can calculate the value of the refine method further, we use the following two metrics.

The distance is the distance from the initial state of the FSM to the final state, i.e., the number of the relevant objects generated between the initial state and the final state is the number of the relevant objects generated between the initial state and the final state.

The difference between the transition times of the path under the pure method and that of the path under the guided method is no less than that of the pure method. For 10 out of 18 cases (55.6%), our guiding method has a higher percentage in relevant paths. For the pure method, for 10 out of 18 cases (55.6%), our guiding method outperforms the pure method significantly with respect to the needed itera-

During analyses, in principle, we want to explore the distribution of the relevant paths explored both many relevant and redundant paths are explored, and paths earlier. Hence, we select the combinations in which the property is a reachability
Paths

<table>
<thead>
<tr>
<th>Time(s)</th>
<th>Apt</th>
<th>Rel</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>49.0%</td>
<td>91.55%</td>
</tr>
<tr>
<td>10.0</td>
<td>49.0%</td>
<td>91.55%</td>
</tr>
<tr>
<td>15.0</td>
<td>49.0%</td>
<td>91.55%</td>
</tr>
<tr>
<td>20.0</td>
<td>49.0%</td>
<td>91.55%</td>
</tr>
</tbody>
</table>

**Figure 4: Relevant path distribution**

- **Value(i)** = \( \sum_{c \in \text{Combinations}} ST_g^c(i) - \sum_{c \in \text{Combinations}} ST_p^c(i) \)

**State Transition Difference**

- The "State transition difference" is calculated as the difference between the number of state transitions in the guided method and the pure method for each path.

**Guided Method vs Pure Method**

- The guided method explores relevant paths more consistently than the pure method, as shown by the higher percentage of relevant paths in the guided method compared to the pure method.

**Figure 5**

- The X-axis represents the path number, and the Y-axis represents the state transition difference.

**Legend**

- "Guided" indicates paths explored under the guided method.
- "Pure" indicates paths explored under the pure method.

- The percentage values in the middle of the graph represent the comparison between the guided and pure methods.
State Transition Difference

Guided DSE causes more state transitions

Value(i) = \sum_{c \in \text{Combinations}} ST_g^c(i) - \sum_{c \in \text{Combinations}} ST_p^c(i)
Conclusion

DSE needs guiding

Key Idea

How about a Regular Property?

Procedure

Evaluate a branch based on its history and future behaviors
Conclusion

- Next step: multi-objects properties, combination with slicing, applications…

DSE needs guiding

How about a Regular Property?

Key Idea

Procedure

Evaluate a branch based on its history and future behaviors.
Thank you
Any Questions?