DART: Directed Automated Random Testing

Patrice Godefroid, Nils Klarlund, Koushik Sen

Presented by:
Geri Grolinger

Instructor:
Professor Azadeh Farzan

Testing

- Primary way to test correctness of the software
- Costs software industry billions of dollars
- 50% of the cost of software development

- Software failures cost US economy about $60 billion a year
- Improvement in testing might save one third of that cost

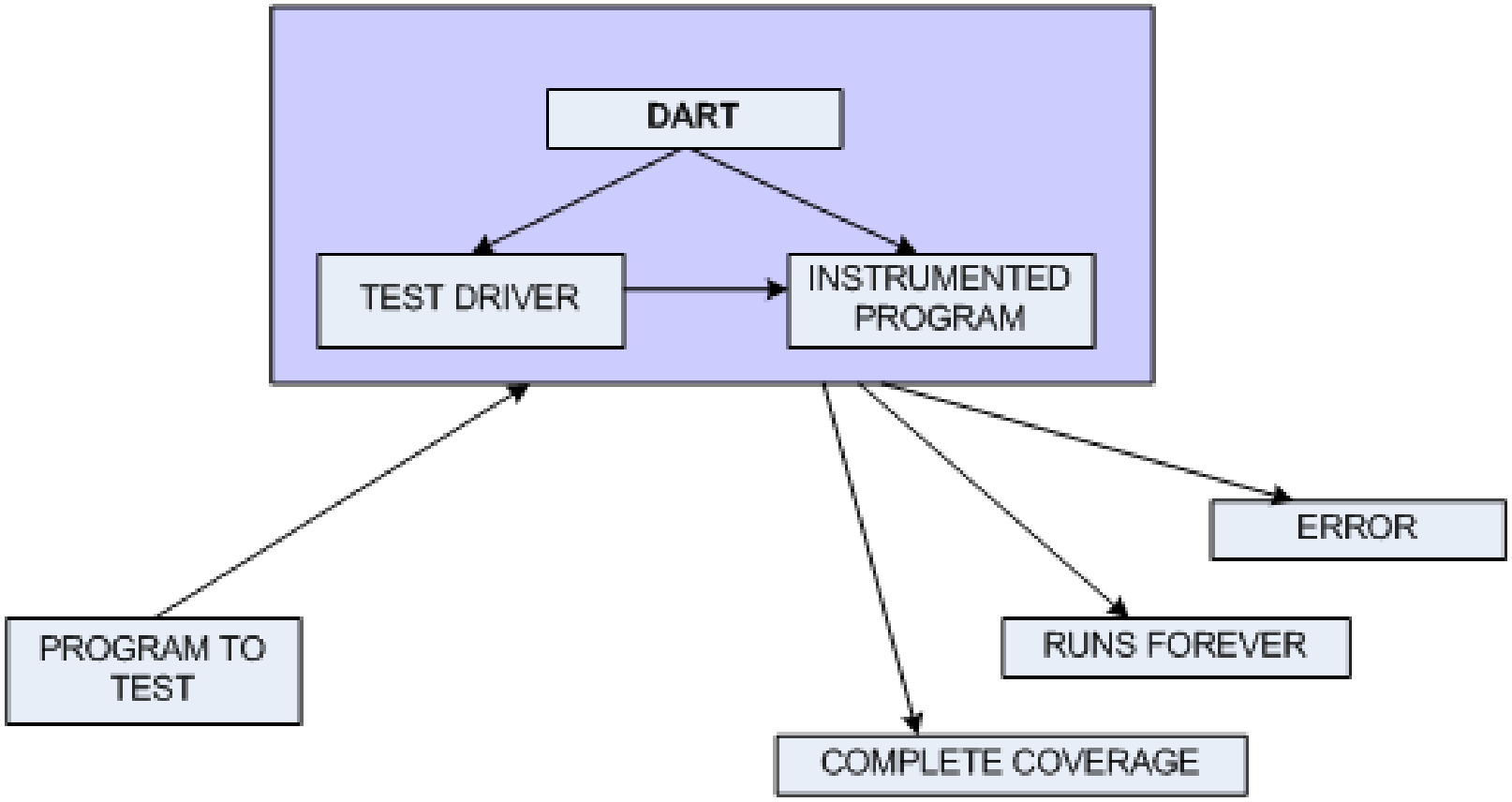
Unit testing

- In theory:
  - Tests individual components of software system
  - Check logic, corner cases etc.
  - Provides 100% code coverage
- In practice:
  - Hard and expensive
  - Requires writing test drivers
  - Rarely done properly

DART automates unit testing

- Combines 3 techniques:
  - Automated interface extraction
  - Automatic generation of a test driver for random testing through the interface
  - Dynamic test generation to direct execution along alternative program paths

DART overview



Program Instrumentation

- Using static code parsing
- **Concrete execution** - original program
- **Symbolic calculations** – original program with interleaved gathering of symbolic constraints

Static code parsing

- Parse code and automatically find:
  - Inputs to the program: arguments to the main function
  - Variables whose values depend on the environment
  - External function calls

Example: Static code parsing

```
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();    /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Testing:
function(int x, int y)

Static parsing finds:
int x
int y

Concrete and symbolic execution

- Concrete execution
- Symbolic execution
  - Original program interleaved with gathering of symbolic constraints
  - Path constraint – input vector that drives program through the current path
  - Path constraints solved (constraint solver) to get the next run to execute the unexplored branch
  - When stuck - falls back to concrete values from concrete execution

Example: Concrete and symbolic execution

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 23, y = 100, temp = 46

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

2 * x != y

Solution to 2 * x = y
Is: x = 5, y = 10

Test Driver

- Combines **random testing** and **directed search**
- Tries to explore all execution paths (directed) while starting with random values

Random testing and directed search

- Random testing
  - Initializes all external variables with random input
- Directed search
  - During each execution, an input vector for next execution is calculated – solution to this symbolic constraint is used as the new input
  - Loops till all execution paths visited or bug found

Example: random test driver, directed search

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 23, y = 100, temp = 46

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

2 * x != in2

Solution to 2 * x = y
Is: x = 5, y = 10

Soundness and completeness

- Sound (in respect to errors found)
  - No false positives among reported errors
- Complete (in a way)
  - If tests terminates without reporting a bug, then no bug exists and all paths are exercised
  - Test driver can run forever

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 12, y = 100

Symbolic execution

symbolic variables x, y

Path constraint

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 23, y = 100, temp = 46

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 23, y = 100, temp = 46

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

2 * x != y

Solution to 2 * x = in2
Is: x = 5, y = 10

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 5, y = 10

Symbolic execution

symbolic variables x, y

Path constraint

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 5, y = 10, temp = 10

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 5, y = 10, temp = 10

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

2 * x == y

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 5, y = 10, temp = 10

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

2 * x == y and
y != x + 10

Solution to 2 * x = y and
y = x + 10
is x = 10, y = 20

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 10, y = 20

Symbolic execution

symbolic variables x, y

Path constraint

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 10, y = 20, temp = 20

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 10, y = 20, temp = 20

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

2 * x == y

Example

```
 1 void main() {
 2     int x = random();
 3     int y = random();
 4
 5     function(x, y);
 6 }
 7
 8 int double(int a) {
 9     return 2*a;
10 }
11
12 void function(int x, int y) {
13
14     int temp = double(x);
15
16     if(temp == y) {
17
18         if(y == x + 10) {
19
20             abort();     /*ERROR*/
21
22         }
23
24     }
25
26 }
```

Concrete execution

x = 10, y = 20, temp = 20

Symbolic execution

symbolic variables x, y,
temp = 2 * x

Path constraint

2 * x == y and
y == x + 10

ERROR FOUND

Dart for C

- Automated interface extraction
  - Program functions, external functions, library functions
- Automated generation of test driver
  - Random initialization of top-level arguments
  - Code for simulating external functions
- Directed search
  - Code instrumentation: CIL
  - Constraint solver: lp_solve

Results: AC-controller

- Toy-program
  - Input filtering
- DART vs random search
  - DART finds errors in less then 1sec (7 runs)
  - Random search runs forever

Results: Needham-Schroeder Protocol

- C implementation of NS public key authentication protocol
- 400 lines of code

- Finds a partial attack in 2 sec (664 runs)
- Finds a full attack in 18 min (328 459 runs)

- DART also found a new bug !

Results: oSIP

- Open source Session Initiation Protocol Library
- 30 000 lines of C code, 600 external functions
- DART crashes 65% of functions in 1000 runs
  - Many due to null-pointer exceptions
- Analysis reveals serious security vulnerability

Conclusion

- DART
  - automates unit testing
  - requires no manually written driver code
  - can test any program that compiles

  - symbolic reasoning in parallel with real execution
  - randomization used where symbolic reasoning is hard

  - improves code coverage vs. pure random testing
  - no false alarms

End

Thanks for listening !