JSConTest
Contract-Driven Testing of JavaScript Code

Phillip Heidegger, Peter Thiemann

Albert-Ludwigs-Universität Freiburg

01.07.2010

Introduction

JavaScript is the language of the Web

Introduction

JavaScript is the language of the Web

99.3%

of all websites use JavaScript

(http://w3techs.com)

Introduction

JavaScript is the language of the Web

99.3%

of all websites use JavaScript

(http://w3techs.com)

How do we ensure that they work correctly and reliably?

Unit Testing

What do we want to test?

- Does the program crash?
- Does the program behave as intended by the programmer?
  (expressed by contracts)

Contributions

- Contract language for JavaScript
- Random testing based on contracts
- Guided random testing to improve coverage
- Contract monitoring
- Implemented in the JSConTest tool

Simple Type Contracts

```
/** int → int */
function f(x) { return 2 * x; };

/** (int,int) → bool */
function p(x,y) {
   if (x != y) {
      if (f(x) == x + 10) return ”false”; // contract violation
   };
   return true;
};
```

Random Testing with Contracts

- Type signature-like contracts
- Type contract in argument position:
  ⇒ random generator
- Type contract in result position:
  ⇒ contract checker

Contract Demo

Demo - ex1.html

Distribution of Test Values

```
/** (int,int) → bool */
function p(x,y) {
  if (x != y) {
    if (f(x) == x + 10) return ”false”; // contract violation
  };
  return true;
};
```

Distribution of Test Values

```
/** (int,int) → bool */
function p(x,y) {
  if (x != y) {
    if (f(x) == x + 10) return ”false”; // contract violation
  };
  return true;
};
```

- random generator for int uniformly distributed

⇒ $P(x = 10) \approx 2^{-32}$

⇒ uniformly distributed generators are not always a good choice

Guided Contract

```
/** (int@numbers,int) → bool */
function p(x,y) {
   if (x != y) {
      if (f(x) == x + 10) return ”false”; // contract violation
   };
   return true;
};
```

- annotate the int contract with @numbers.

Guided Contract

```
/** (int@numbers,int) → bool */
function p(x,y) {
  if (x != y) {
    if (f(x) == x + 10) return "false"; // contract violation
  };
  return true;
};
```

- annotate the int contract with @numbers.

⇒ Changes the probability distribution

⇒ Generates random expressions with numbers from the source program

⇒ Usually locates the violation in less than 10 test runs

Guided Contract – Demo

Demo - ex1a.html

Guided Contract

```
/** (int@numbers,int@numbers,int@numbers) → bool */
function fut_1(x,y,z) {
  if ((x*3 + 5 == y*5 + 4) && (x*2 − 1 == z*9 − 1))
    return ”false”; // contract violation
  return true;
};
```

Guided Contract

```
/** (int@numbers,int@numbers,int@numbers) → bool */
function fut_1(x,y,z) {
  if ((x*3 + 5 == y*5 + 4) && (x*2 − 1 == z*9 − 1))
    return ”false”; // contract violation
  return true;
};
```

- complex conditional → difficult to archive high coverage
- Our approach detects the violation in less than 5 sec

Guided Contract for Objects

```
/** (object) → bool */
function h(x) {
  if (x && x.p && x.quest)
    return ”false”; // contract violation
  return true;
};
```

- Blindly generating random objects does not lead to high coverage
- How to guide the random generator for objects?

Guided Contract for Objects

```
/** (object@labels) → bool */
function h(x) {
  if (x && x.p && x.quest)
    return ”false”; // contract violation
  return true;
};
```

- Annotation @labels
- Generator prefers to use the labels inside of the function body

⇒ Raises probability to generate a property with names p or quest

Contract Monitoring

```
/∗∗ int → int ∗/
function f(x) { return 2 ∗ x; };

/∗∗ (int,int) → bool ∗/
function g(x,y) {
   return (f(x ∗ ”3O”) == 60);
}
```

- Where is the bug?

Contract Monitoring

```
/** int → int */
function f(x) { return 2 * x; };

/** (int,int) → bool */
function g(x,y) {
  return (f(x * ”3O”) == 60);
}
```

- Where is the bug?

→ Programmer wrote O instead of zero.

- Does not violate the contract of g.
- But violates the contract of f.

Contract Monitoring

- JSConTest generates assertions for checking argument and result contracts (pre- and postcondition)
- If during a run of `g`, the contract of `f` is violated, the assertions report this violation

Huffman Encoding

- Take textbook algorithm
- Specify the behavior of the code
- Custom contract for Huffman Trees (13 loc)
- Annotations to functions (3 loc)

Huffman Encoding

- After contract specification we found
  - a typing error in our code
  - a bug inside of the contract specification
- To check the effectivity of contract checking:
  We applied mutators to the Huffman Code
  - 88% of the mutated programs were rejected
  - 12% pass
    - manual inspection of the 12% shows that they behave correct with respect to the (type) specification

  ⇒ JSConTest detects type errors reliably

Related Work

- K. Claessen, J. Hughes, QuickCheck, ICFP 2000
- C. Csallner, Y. Smaragdakis, JCrasher, SPE 2004
- Guha, Matthews, Findler, Krishnamurthi, Relationally-Parametric Polymorphic Contracts, DLS 2007

Conclusion

- Contract language for JavaScript
- Random testing and contract monitoring
- Guided random testing to improve coverage
- Implemented in the JSConTest tool

Conclusion

- Contract language for JavaScript
- Random testing and contract monitoring
- Guided random testing to improve coverage
- Implemented in the JSConTest tool

Future Work

- Minimization of counterexamples
- Transactions for JavaScript (Side Effects)
- Extension of contract language to describe side effects