TreatJS

Higher-Order Contracts for JavaScript

UNI FREIBURG

Roman Matthias Keil, Peter Thiemann
University of Freiburg, Germany

July 8, 2015, The European Conference on Object-Oriented Programming, ECOOP 2015
Prague, Czech Republic

Notizen

Introduction

UNI FREIBURG

Contract

- Specifies the interface of a software component
- Obligations - Benefits

TreatJS

- Language embedded contract system for JavaScript
- Enforced by run-time monitoring
- Inspired by existing contract systems

Notizen

Features

UNI FREIBURG

- Standard abstractions for higher-order-contracts (base, function, and dependent contracts) [Findler,Felleisen'02]
- Intersection and union contracts
- Side-effect free contract execution
- Contract constructors generalize dependent contracts

Notizen

Base Contract [Findler,Felleisen'02]

- *Base Contracts* are built from predicates
- Specified by a plain JavaScript function

```
1 function isNumber (arg) {
2   return (typeof arg) === 'number';
3 };
4 var _Number_ = Contract.Base(isNumber);

5 assert(1, _Number_); ✓
6 assert('a', _Number_); ✗ blame the subject
```

- Subject v *gets blamed* for *Base Contract* $\mathcal{B}$ iff: $\mathcal{B}(v) \neq true$

Notizen

Function Contract [Findler,Felleisen'02]

```
1 // Number × Number → Number
2 function plus (x, y) {
3   return (x + y);
4 }

5 var plus = assert(plus,
6   Contract.Function([_Number_, _Number_], _Number_));
```


Notizen

Function Contract [Findler,Felleisen'02]

```
1 // Number × Number → Number
2 function plus (x, y) {
3   return (x + y);
4 }

5 plus('a', 'a'); ✗ blame the context
```

- Context gets *blamed* for $\mathcal{C} \to \mathcal{C}'$ iff: Argument x gets *blamed* for $\mathcal{C}$ (as subject)

Notizen

Function Contract [Findler,Felleisen'02]

```
1 // Number × Number → Number
2 function plusBroken (x, y) {
3   return (x>0 && y>0) ? (x + y) : 'Error';
4 }
5 plusBroken(0, 1); ✗ blame the subject
```

- Subject f gets *blamed* for $\mathcal{C} \to \mathcal{C}'$ iff:
  $\neg$ (Context gets *blamed* $\mathcal{C}$) $\wedge$ ($f(x)$ gets *blamed* $\mathcal{C}'$)

Notizen

TreatJS

New!

Notizen

Overloaded Operator

- Function *plus* works for strings, too
- Requies to model overloading and multiple inheritances

```
1 // Number × Number → Number
2 function plus (x, y) {
3   return (x + y);
4 }
5 plus('a', 'a'); ✗ blame the context
```


Notizen

Intersection Contract


```
1  // (Number × Number → Number) ∩ (String × String → String)
2  function plus (x, y) {
3    return (x + y);
4  }

5  var plus = assert(plus, Contract.Intersection(
6    Contract.Function([_Number_, _Number_], _Number_)
7    Contract.Function([_String_, _String_], _String_));
```


Notizen

Intersection Contract


```
1  // (Number × Number → Number) ∩ (String × String → String)
2  function plus (x, y) {
3    return (x + y);
4  }

5  plus(true, true); ✗ blame the context
```

- Context gets *blamed* for $\mathcal{C} \cap \mathcal{C}'$ iff:
  (Context gets *blamed* for $\mathcal{C}$) $\wedge$ (Context gets *blamed* for $\mathcal{C}'$)

Notizen

Intersection Contract


```
1  // (Number × Number → Number) ∩ (String × String → String)
2  function plusBroken (x, y) {
3    return (x>0 && y>0) ? (x + y) : 'Error';
4  }

5  plusBroken(0, 1); ✗ blame the subject
```

- Subject f gets *blamed* for $\mathcal{C} \cap \mathcal{C}'$ iff:
  (f gets *blamed* for $\mathcal{C}$) $\vee$ (f gets *blamed* for $\mathcal{C}'$)

Notizen

Union Contract

```
1  // (Number × Number → Number) ∪ (Number × Number → String)
2  function plusBroken (x, y) {
3    return (x>0 && y>0) ? (x + y) : 'Error';
4  }

5  var plusBroken = assert(plusBroken, Contract.Union(
6    Contract.Function([_Number_, _Number_], _Number_)
7    Contract.Function([_Number_, _Number_], _String_));
```


Notizen

Union Contract

```
1  // (Number × Number → Number) ∪ (Number × Number → String)
2  function plusBroken (x, y) {
3    return (x>0 && y>0) ? (x + y) : 'Error';
4  }

5  plusBroken('a', 'a'); ✗ blame the context
```

- Context gets *blamed* for $\mathcal{C} \cup \mathcal{C}'$ iff:
  (Context gets *blamed* for $\mathcal{C}$) $\vee$ (Context gets *blamed* for $\mathcal{C}'$)

Notizen

Union Contract

```
1  // (Number × Number → Number) ∪ (Number × Number → String)
2  function plusBroken (x, y) {
3    return (x>0 && y>0) ? (x + y) : 'Error';
4  }

5  plusBroken(1, 1); ✓
6  plusBroken(0, 1); ✗ blame the subject
```

- Subject f gets *blamed* for $\mathcal{C} \cup \mathcal{C}'$ iff:
  (f gets *blamed* for $\mathcal{C}$) $\wedge$ (f gets *blamed* for $\mathcal{C}'$)

Notizen

Contract Assertion

- A failing contract must not signal a violation immediately
- Violation depends on combinations of failures in different sub-contracts

```
1 (Number → Number) ∩ (String → String)
2 function addOne (x) {
3   return (x + 1);
4 }
```

```
1 addOne('a');
```


Notizen

Contract Assertion

- A failing contract must not signal a violation immediately
- Violation depends on combinations of failures in different sub-contracts

```
1 (Number → Number) ∩ (String → String)
2 function addOne (x) {
3   return (x + 1);
4 }
```

```
2 addOne('a');
```


Notizen

Contract Assertion

- A failing contract must not signal a violation immediately
- Violation depends on combinations of failures in different sub-contracts

```
1 (Number → Number) ∩ (String → String)
2 function addOne (x) {
3   return (x + 1);
4 }
```

```
3 addOne('a'); ✓
```


Notizen

Blame Calculation

- Contract assertion must connect each contract with the enclosing operations
- *Callback* implements a constraint and links each contracts to its next enclosing operation
- Reports a record containing two fields, *context* and *subject*
- Fields range over $\mathbb{B}_4 = \{\bot, \mathsf{f}, \mathsf{t}, \top\}$ [Belnap'1977]

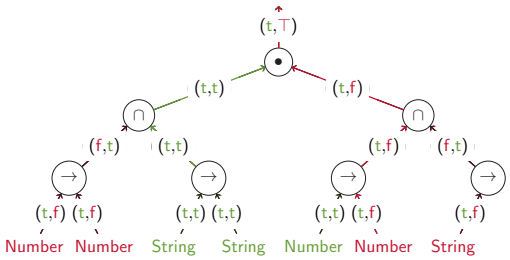
Notizen

Callback Graph

```
1 (Number → Number) ∩ (String → String)
2 function addOneBroken (x) {
3   return (x + '1');
4 }
5 addOneBroken('a'); ✓
6 addOneBroken(1); ✗ blame the subject
```


Notizen

Callback Graph




```
1 (Number → Number) ∩ (String → String)
2 addOneBroken('a'1); ✓blame the subject
```


Notizen

Non-Interference

- No syntactic restrictions on predicates
- Problem: Contract may interfere with program execution
- Solution: Predicate evaluation takes place in a sandbox

```
1 function isNumber (arg) {
2   type = (typeof arg);
3   return type === 'number';
4 };
5 var _Number_ = Contract.Base(isNumber);
```


Notizen

Non-Interference

- No syntactic restrictions on predicates
- Problem: Contract may interfere with program execution
- Solution: Predicate evaluation takes place in a sandbox

```
1 function isNumber (arg) {
2   type = (typeof arg); ✗ access forbidden
3   return type === 'number';
4 };
5 var _Number_ = Contract.Base(isNumber);
6 assert(1, _Number_);
```


Notizen

Sandbox

- All contracts guarantee noninterference
- Read-only access is safe

```
1 var _Array_ = Contract.Base(function (arg) {
2   return (arg instanceof Array); ✗ access forbidden
3 });
```


Notizen

Sandbox

- All contracts guarantee noninterference
- Read-only access is safe

```
1 var _Array_ = Contract.Base(function (arg) {
2   return (arg instanceof OutsideArray); ✓
3 });
4 var _Array_ = Contract.With({OutsideArray:Array}, _Array_);
```


Notizen

Contract Constructor

- Building block for dependent, parameterized, abstract, and recursive contracts
- Constructor gets evaluated in a sandbox, like a predicate
- Returns a contract
- No further sandboxing for predicates

```
1 var __Type__ = Contract.Constructor(function (type) {
2   return Contract.Base(function (arg) {
3     return (typeof arg) === type;
4   });
5 });
6 var _Number_ = __Type__('number');
```


Notizen

Contract Abstraction


```
1 // T × T → T
2 function plus (x, y) {
3   return (x + y);
4 }
5 var __Plus__ = Contract.Constructor(function (_Type_) {
6   return Contract.Function([_Type_, _Type_], _Type_);
7 });
8 var Plus = assert(plus, __Plus__);
9 Plus(_Number_)(1, 2); ✓
```


Notizen

Dependent Contract

```
1  // T × T → T
2  function plus (x, y) {
3    return (x + y);
4  }

5  var __Type__ = Contract.Constructor(function(x, y) {
6    return Contract.Base(function (arg) {
7      return ((typeof x) === (typeof y)) &&
8        ((typeof x) === (typeof arg));
9    });
10 });

11 var plus = assert(plus, Contract.Dependent(__Type__));

12 plus(1, 2); ✓
```


Notizen

Conclusion

- TreatJS: Language embedded, dynamic, higher-order contract system for full JavaScript
- Support for intersection and union contracts
- Systematic blame calculation
- Composable sandboxing that guarantees non-interference
- Contract constructors with local scope

Notizen

Notizen