

# TreatJS

## Higher-Order Contracts for JavaScript



UNI  
FREIBURG

*Matthias Keil, Peter Thiemann*

Institute for Computer Science  
University of Freiburg  
Freiburg, Germany

June 30, 2014, Dagstuhl Seminar on Scripting Languages and Frameworks: Analysis and Verification

## TreatJS

- Language embedded contract system for JavaScript
- Enforced by run-time monitoring
- Inspiration similar to *Contracts.js* [Disney]

## Contract

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

- Standard abstractions for higher-order-contracts (base, function, and object contracts) [Findler,Felleisen'02]
- Support for boolean combinations (and, or, not contracts) as building blocks for intersection, union, and implication
- Contract constructors generalize dependent contracts
- Non-interference for contract execution



# Base Contract [Findler,Felleisen'02]

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

```
1 function typeOfNumber (arg) {  
2   return (typeof arg) === 'number';  
3 };  
4 var IsNumber = BaseContract(typeOfNumber, 'IsNumber');
```

# Base Contract [Findler, Felleisen '02]

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

```
1 function typeOfNumber (arg) {  
2   return (typeof arg) === 'number';  
3 };  
4 var IsNumber = BaseContract(typeOfNumber, 'IsNumber');  
5 assert(1, IsNumber); ✓
```

- Value  $v$  fulfills **BaseContract** ( $\mathcal{B}$ ) iff:  $\mathcal{B}(v) = true$

# Base Contract [Findler,Felleisen'02]

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

```
1 function typeOfNumber (arg) {  
2   return (typeof arg) === 'number';  
3 };  
4 var IsNumber = BaseContract(typeOfNumber, 'IsNumber');  
5 assert('a', IsNumber); X Blame the Value
```

- Value  $v$  gets blamed for contract  $\mathcal{B}$  iff:  $\mathcal{B}(v) = \text{false}$

# Function Contract [Findler,Felleisen'02]



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



```
1 // Number × Number → Number
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5
6 var addChecked = assert(addUnchecked,
7   FunctionContract([IsNumber, IsNumber], IsNumber));
```



# Function Contract [Findler,Felleisen'02]

```

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

- $x$  is an *acceptable argument for contract*  $\mathcal{C} \rightarrow \mathcal{C}'$  iff:  
( $x$  fulfills  $\mathcal{C}$ )
- Function  $f$  fulfills contract  $\mathcal{C} \rightarrow \mathcal{C}'$  at argument  $x$  iff:  
( $x$  fulfills  $\mathcal{C}$ )  $\rightarrow$  ( $f(x)$  fulfills  $\mathcal{C}'$ )

# Function Contract [Findler,Felleisen'02]

```
1 // Number × Number → Number
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5 addChecked('a', 'a'); X Blame the Argument
```

- Argument  $x$  gets blamed for  $C \rightarrow C'$  iff:
  - $\neg$  ( $x$  is an *acceptable argument* for contract  $C \rightarrow C'$ ) iff:
  - $\neg$  ( $x$  fulfills  $C$ )

# Function Contract [Findler,Felleisen'02]

```

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

- Function  $f$  gets blamed for  $\mathcal{C} \rightarrow \mathcal{C}'$  at argument  $x$  iff:  
 $\neg$  (Function  $f$  fulfills contract  $\mathcal{C} \rightarrow \mathcal{C}'$  at argument  $x$ ) iff:  
 $(x \text{ fulfills } \mathcal{C}) \wedge \neg (f(x) \text{ fulfills } \mathcal{C}')$



# New!



# Intersection Contract

```
1 // Number × Number → Number
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5 addChecked('a', 'a'); X
```

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

```
1 // (Number × Number → Number) ∩ (String × String → String)
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5 var addChecked = assert(addUnchecked, Intersection(
6   FunctionContract([IsNumber, IsNumber], IsNumber)
7   FunctionContract([IsString, IsString], IsString));
```

# Intersection Contract

```

1 // (Number × Number → Number) ∩ (String × String → String)
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5 addChecked('a', 'a'); ✓
  
```

- $x$  is an acceptable argument for contract  $\mathcal{C} \cap \mathcal{C}'$  iff:  
 $(x \text{ is acceptable arg. for } \mathcal{C}) \vee (x \text{ is acceptable arg. for } \mathcal{C}')$
- Function  $f$  fulfills contract  $\mathcal{C} \cap \mathcal{C}'$  at argument  $x$  iff:  
 $(f \text{ fulfills } \mathcal{C} \text{ at } x) \wedge (f \text{ fulfills } \mathcal{C}' \text{ at } x)$



```
1 // (Number × Number → Number) ∩ (String × String → String)
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5 addChecked(true, true); ✗ Blame the Argument
```

- Argument  $x$  gets blamed for  $\mathcal{C} \cap \mathcal{C}'$  iff:

# Intersection Contract

```
1 // (Number × Number → Number) ∩ (String × String → String)
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5 addChecked(true, true); ✗ Blame the Argument
```

- Argument  $x$  gets blamed for  $\mathcal{C} \cap \mathcal{C}'$  iff:
- $\neg$  ( $x$  is an acceptable argument for contract  $\mathcal{C} \cap \mathcal{C}'$ )

# Intersection Contract

```
1 // (Number × Number → Number) ∩ (String × String → String)
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5 addChecked(true, true); ✗ Blame the Argument
```

- Argument  $x$  gets blamed for  $\mathcal{C} \cap \mathcal{C}'$  iff:
  - $\neg$  ( $x$  is an acceptable argument for contract  $\mathcal{C} \cap \mathcal{C}'$ )
  - $\neg$  (( $x$  is acc. arg. for  $\mathcal{C}$ )  $\vee$  ( $x$  is acc. arg. for  $\mathcal{C}'$ ))

# Intersection Contract

```

1 // (Number × Number → Number) ∩ (String × String → String)
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5 addChecked(true, true); ✗ Blame the Argument
  
```

- Argument  $x$  gets blamed for  $\mathcal{C} \cap \mathcal{C}'$  iff:
  - $\neg$  ( $x$  is an acceptable argument for contract  $\mathcal{C} \cap \mathcal{C}'$ )
  - $\neg$  (( $x$  is acc. arg. for  $\mathcal{C}$ )  $\vee$  ( $x$  is acc. arg. for  $\mathcal{C}'$ ))
  - $\neg$ ( $x$  is acc. arg. for  $\mathcal{C}$ )  $\wedge$   $\neg$ ( $x$  is acc. arg. for  $\mathcal{C}'$ )

# Intersection Contract

```

1 // (Number × Number → Number) ∩ (String × String → String)
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5 addChecked(true, true); ✗ Blame the Argument
  
```

- Argument  $x$  gets blamed for  $\mathcal{C} \cap \mathcal{C}'$  iff:
  - $\neg$  ( $x$  is an acceptable argument for contract  $\mathcal{C} \cap \mathcal{C}'$ )
  - $\neg$  (( $x$  is acc. arg. for  $\mathcal{C}$ )  $\vee$  ( $x$  is acc. arg. for  $\mathcal{C}'$ ))
  - $\neg$ ( $x$  is acc. arg. for  $\mathcal{C}$ )  $\wedge$   $\neg$ ( $x$  is acc. arg. for  $\mathcal{C}'$ )
  - ( $x$  gets blamed for  $\mathcal{C}$ )  $\wedge$  ( $x$  gets blamed for  $\mathcal{C}'$ )

# Intersection Contract

```
1 // (Number × Number → Number) ∩ (String × String → String)
2 function addUnchecked(x, y) {
3   return (x>0 && y>0) ? (x + y) : 'Error';
4 }
5 addChecked(0, 1); ✗ Blame the Function
```

- Function  $f$  gets blamed for  $\mathcal{C} \cap \mathcal{C}'$  at argument  $x$  iff:  
( $f$  gets blamed for  $\mathcal{C}$  at  $x$ )  $\vee$  ( $f$  gets blamed for  $\mathcal{C}'$  at  $x$ )

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

```
1 // (Number × Number → Number) ∪ (Number × Number → String)
2 function addUnchecked(x, y) {
3   return (x>0 && y>0) ? (x + y) : 'Error';
4 }
5 var addChecked = assert(addUnchecked, Union(
6   FunctionContract([IsNumber, IsNumber], IsNumber)
7   FunctionContract([IsNumber, IsNumber], IsString));
```



# Union Contract

```

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

- Argument  $x$  is an acceptable argument for contract  $\mathcal{C} \cup \mathcal{C}'$  iff:  
 $(x \text{ is acceptable arg. for } \mathcal{C}) \wedge (x \text{ is acceptable arg. for } \mathcal{C}')$
- Function  $f$  fulfills contract  $\mathcal{C} \cup \mathcal{C}'$  at argument  $x$  iff:  
 $(f \text{ fulfills } \mathcal{C} \text{ at } x) \vee (f \text{ fulfills } \mathcal{C}' \text{ at } x)$

```
1 // (Number × Number → Number) ∪ (Number × Number → String)
2 function addUnchecked(x, y) {
3   return (x>0 && y>0) ? (x + y) : 'Error';
4 }
5 addChecked('a', 'a'); X Blame the Argument
```

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

```
1 // (Number × Number → Number) ∪ (Number × Number → String)
2 function addUnchecked(x, y) {
3   return (x>0 && y>0) ? (x + y) : undefined;
4 }
5 addChecked(0, 1); ✗ Blame the Function
```

- Function  $f$  gets blamed for  $\mathcal{C} \cup \mathcal{C}'$  at argument  $x$  iff:  
( $f$  gets blamed for  $\mathcal{C}$  at  $x$ )  $\wedge$  ( $f$  gets blamed for  $\mathcal{C}'$ ) at  $x$

# Non-Interference

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

```
1 function typeOfNumber (arg) {  
2   type = (typeof arg); X Access forbidden  
3   return type === 'number';  
4 };  
5 var FaultyIsNumber =  
6   BaseContract(typeOfNumber, 'FaultyIsNumber');
```

# Non-Interference

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

```
1 var isArray = BaseContract(function (arg) {  
2   return (arg instanceof Array); X Access forbidden  
3 });
```

# Non-Interference

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

```
1 var isArray = BaseContract(function (arg) {  
2   return (arg instanceof InsideArray); ✓  
3 });  
4 var isArrayComplete = With({InsideArray:Array}, isArray);
```

# Contract Constructor



```
1 //  $T \times T \rightarrow T$   
2 function addUnchecked(x, y) {  
3   return (x + y);  
4 }
```

```
1 //  $T \times T \rightarrow T$ 
2 function addUnchecked(x, y) {
3   return (x + y);
4 }
5
6 var addChecked = assert(addUnchecked,
7   FunctionContract([CheckType, CheckType], CheckType)):
8
9 var CheckType = BaseContract(function (arg) {
10   // to be completed
11 });
```



- Constructor gets evaluated in a sandbox, like a predicate
- Returns a contract
- No further sandboxing for predicates

```
1 var ctor = Constructor(function() {  
2   var type = undefined;  
3   var CheckType = BaseContract(function (arg) {  
4     type = type || (typeof arg);  
5     return type === (typeof arg);  
6   });  
7   return FunctionContract([CheckType, CheckType], CheckType):  
8   }, 'SameType');
```

```
1 var SameTypeCtor = Constructor(function(arg) {  
2   var type = (typeof arg);  
3   return BaseContract(function (arg) {  
4     return (typeof arg) === type);  
5   });  
6 });
```

```
1 var SameTypeCtor = Constructor(function(arg) {
2   var type = (typeof arg);
3   return BaseContract(function (arg) {
4     return (typeof arg) === type);
5   });
6 });

7 var addChecked = assert(addUnchecked,
8   DependentContract(SameTypeCtor));
```

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