Introduction

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

Features

- 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 Contracts [Findler,Felleisen'02]

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

```javascript
function typeOfNumber (arg) {
  return (typeof arg) === 'number';
}
var IsNumber = BaseContract(typeOfNumber, 'IsNumber');
assert(1, IsNumber);
```

Value \( v \) fulfills \( B \) iff: \( B(v) = true \)

Notizen

Base Contracts [Findler,Felleisen'02]

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

```javascript
function typeOfNumber (arg) {
  return (typeof arg) === 'number';
}
var IsNumber = BaseContract(typeOfNumber, 'IsNumber');
assert('a', IsNumber);
```

Value \( v \) gets blamed for contract \( B \) iff: \( B(v) = false \)

Notizen

Function Contract [Findler,Felleisen'02]

```javascript
// Number \times Number \rightarrow Number
function addUnchecked(x, y) {
  return (x + y);
}
var addChecked = assert(addUnchecked,
FunctionContract([IsNumber, IsNumber], IsNumber));
```

Notizen
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 C → C’ iff: (x fulfills C)
- Function f fulfills contract C’ → C at argument x iff: (x fulfills C) → (f(x) fulfills C’)

Notizen

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);

- Argument x gets blamed for C’ → C’’ iff: ¬(x is an acceptable argument for contract C → C’’)
  iff: ¬(x fulfills C)

Notizen

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);

- Function f gets blamed for C’ → C’’ at argument x iff: ¬(Function f fulfills contract C’ → C’’ at argument x) iff: (x fulfills C) ∧ ¬(f(x) fulfills C’’)

Notizen
Intersection Contract

1 // Number × Number → Number
2 function addUnchecked(x, y) {
3    return (x + y);
4 }
5
6 addChecked(‘a’, ‘a’);

Intersection Contract

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

Notizen
Intersection Contract

// (Number × Number → Number) ∩ (String × String → String)

function addUnchecked(x, y) {
    return (x + y);
}

addChecked('a', 'a'); ✓

| Matthias Keil, Peter Thiemann | TreatJS | June 30, 2014 | 8 / 14 |

Blame the Argument

Argument x gets blamed for \( C \cap C' \) iff:
- \( \neg (x \text{ is an acceptable argument for } C \cap C') \)
- \( \neg ((x \text{ is acc. arg. for } C) \lor (x \text{ is acc. arg. for } C')) \)
- \( \neg (x \text{ is acc. arg. for } C) \land \neg (x \text{ is acc. arg. for } C') \)
- \( x \text{ gets blamed for } C \land \neg (x \text{ gets blamed for } C) \)

| Matthias Keil, Peter Thiemann | TreatJS | June 30, 2014 | 8 / 14 |

Blame the Function

Function f gets blamed for \( C \cap C' \) at argument x iff:
- \( f \text{ gets blamed for } C \land f \text{ gets blamed for } C' \land f \text{ gets blamed for } C \land f \text{ gets blamed for } C' \)

| Matthias Keil, Peter Thiemann | TreatJS | June 30, 2014 | 8 / 14 |
function addUnchecked(x, y) {
    return (x > 0 && y > 0) ? (x + y) : 'Error';
}

var addChecked = assert(addUnchecked, Union(/
  FunctionContract([IsNumber, IsNumber], IsNumber)

var addChecked = assert(addUnchecked, Union(/
  FunctionContract([IsNumber, IsNumber], IsNumber)

assert(addChecked(0, 1));

// Argument x is an acceptable argument for contract C ∪ C' iff:
// (x is acceptable arg. for C) ∧ (x is acceptable arg. for C')
// (f fulfills C at x) ∨ (f fulfills C' at x)

assert(addChecked('a', 'a')); // Blame the Argument

// Argument x gets blamed for C ∪ C' iff:
// (x gets blamed for C) ∨ (x gets blamed for C')
Union Contract

// (Number x Number -> Number) u (Number x Number -> String)
function addUnchecked(x, y) {
  return (x>0 & & y>0) ? (x + y) : undefined;
}

addChecked(0, 1); // Blame the Function

Non-Interference

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

function typeOfNumber (arg) {
  type = (typeof arg); // Access forbidden
  return type === 'number';
};

var FaultyIsNumber = 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

var isArray = BaseContract/function (arg) {
  return (arg instanceof Array); // Access forbidden
};
Non-Interference

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

```javascript
1 var isArray = BaseContract(function (arg) {
2    return (arg instanceof InsideArray);
3 });
4 var isArrayComplete = With(['InsideArray', 'Array'], isArray);
```

Solution: Predicate evaluation takes place in a sandbox.

Contract Constructor

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

```javascript
1 // T × T → T
2 function addUnchecked(x, y) {
3    return (x + y);
4 }
5 var addChecked = assert(addUnchecked,
6    FunctionContract([CheckType, CheckType], CheckType));
7 var CheckType = BaseContract(function (arg) {
8    // to be completed
9 });
10 return FunctionContract([CheckType, CheckType], CheckType):
11 , 'SameType');
```
Dependent Contract

```javascript
var SameTypeCtor = Constructor(function(arg) {
  var type = (typeof arg);
  return BaseContract(function(arg) {
    return (typeof arg) === type);
  });
});

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

Conclusion

- 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