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

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

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

Base Contract [Findler,Felleisen'02]

UNI FREIBURG

- *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:
  $\mathcal{B}(v) \neq true$

Function Contract [Findler,Felleisen'02]

UNI FREIBURG

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

Function Contract [Findler,Felleisen'02]

UNI FREIBURG

```
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_));
```

Function Contract [Findler,Felleisen'02]

UNI FREIBURG

```
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)

Function Contract [Findler,Felleisen'02]

UNI FREIBURG

```
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* $\mathcal{C}'$)

New!

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
```

Intersection Contract

UNI FREIBURG

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

Intersection Contract

UNI FREIBURG

```
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_));
```

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 $\mathcal{C}'$)

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 $\mathcal{C}'$)

Union Contract

UNI FREIBURG

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

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_));
```

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 $\mathcal{C}'$)

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 $\mathcal{C}'$)

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');
```

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');
```

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'); ✓
```

Blame Calculation

UNI FREIBURG

- 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]

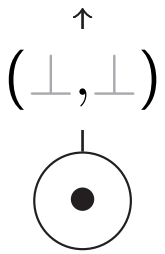
Callback Graph

UNI FREIBURG

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

5  addOneBroken('a'); ✓
6  addOneBroken(1); ✗ blame the subject
```

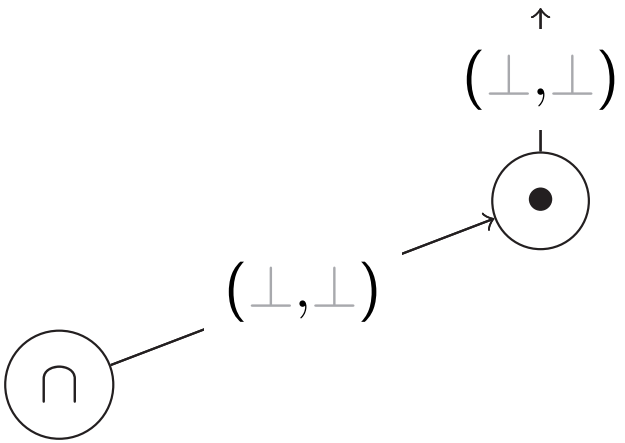
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken('a');*

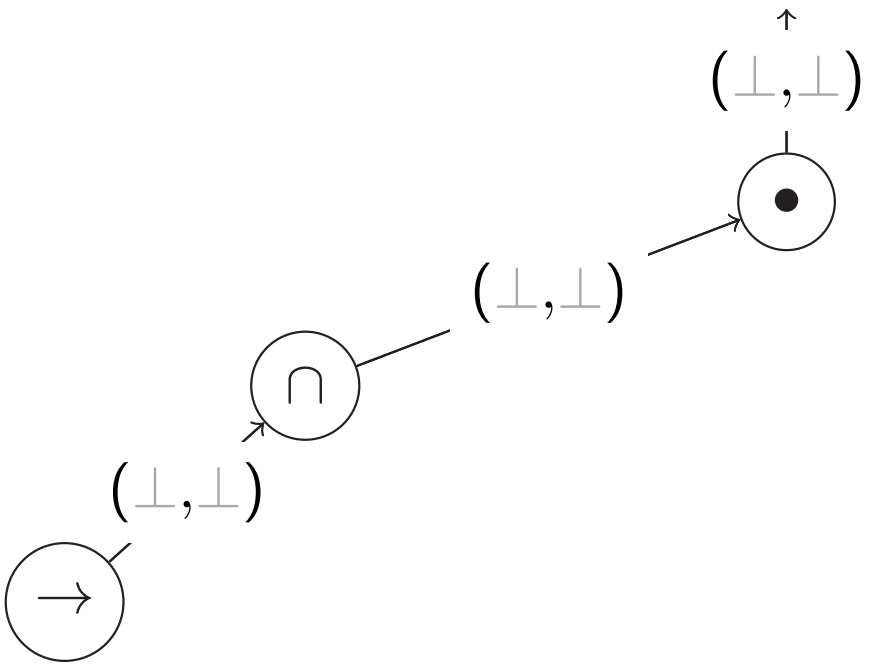
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken('a');*

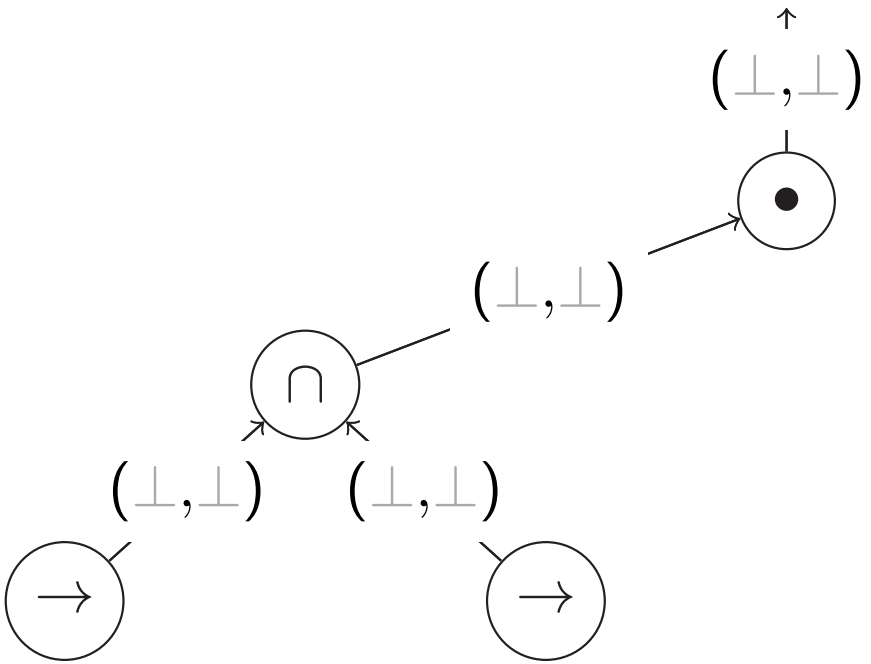
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken('a');*

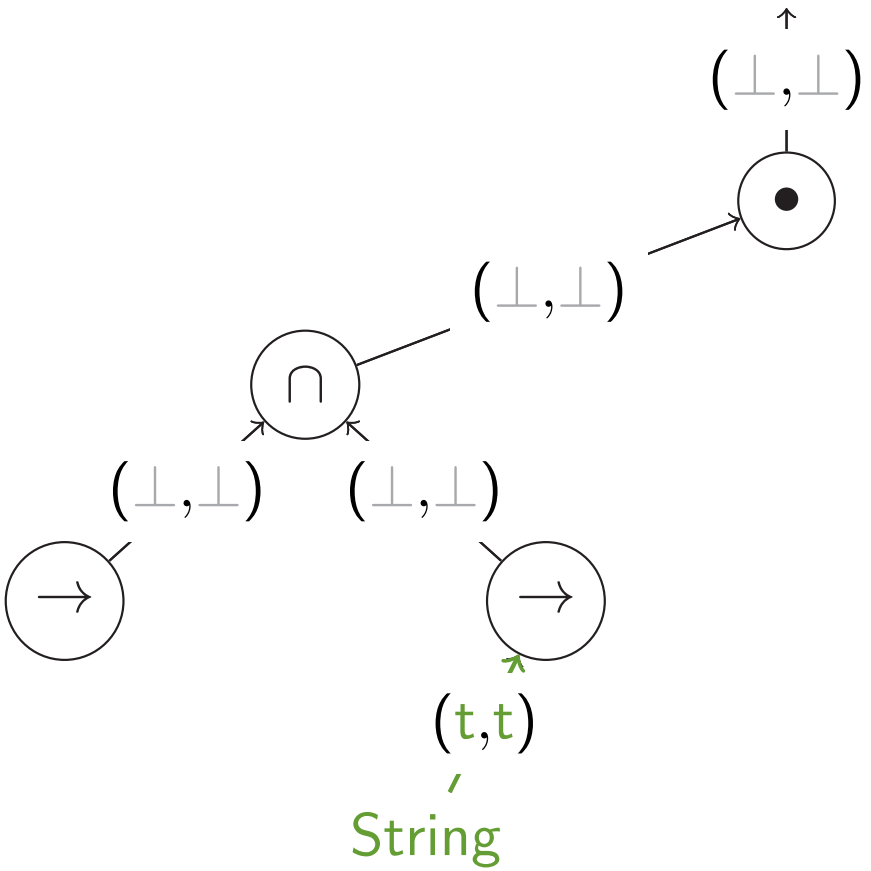
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken('a');*

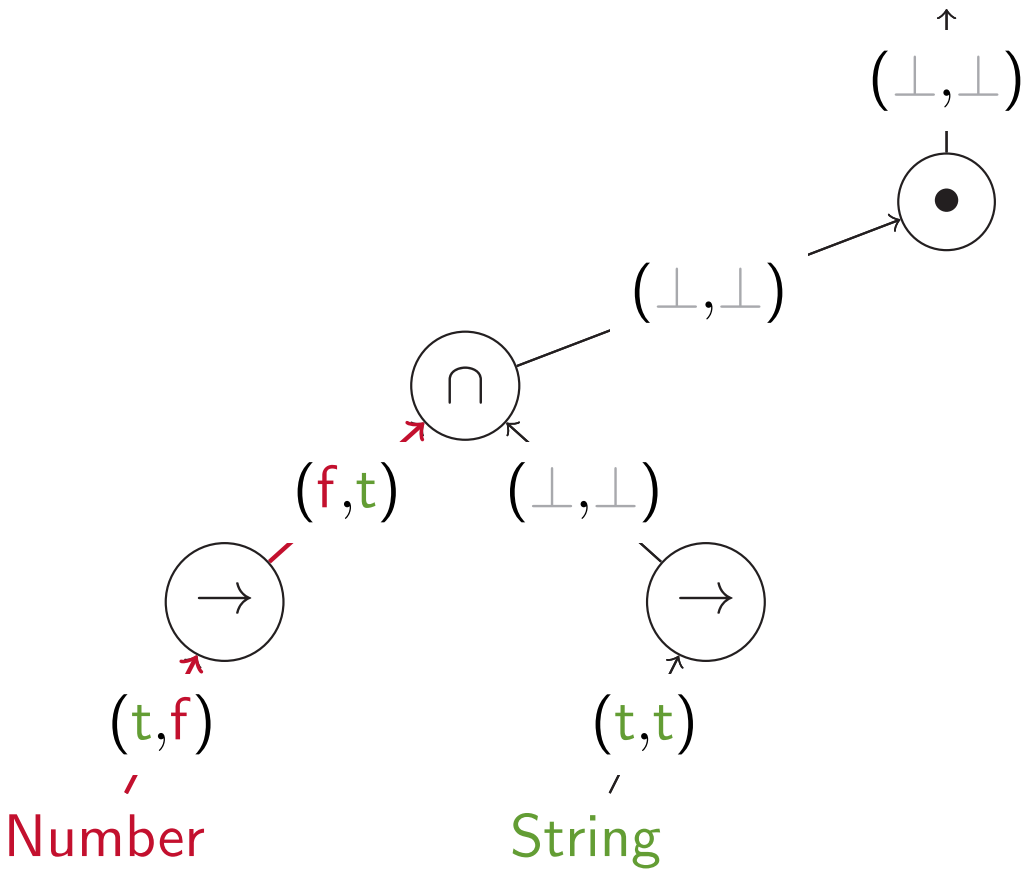
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken('a');*

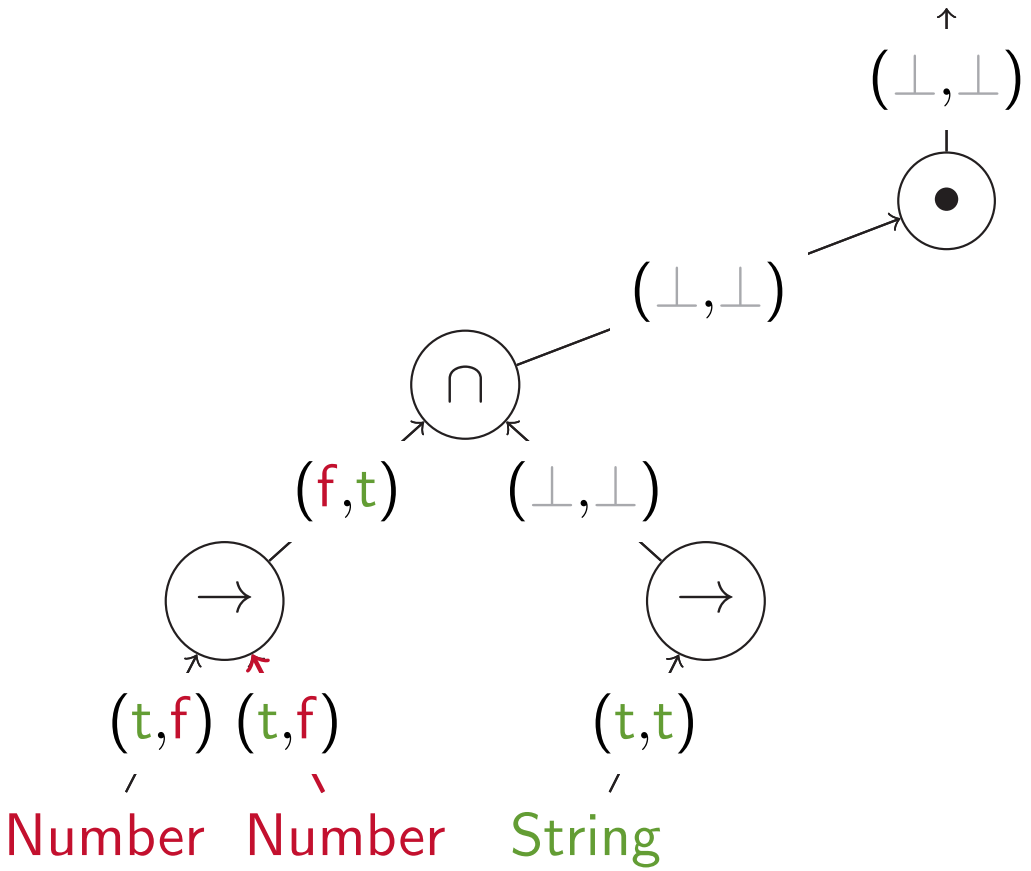
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken('a');*

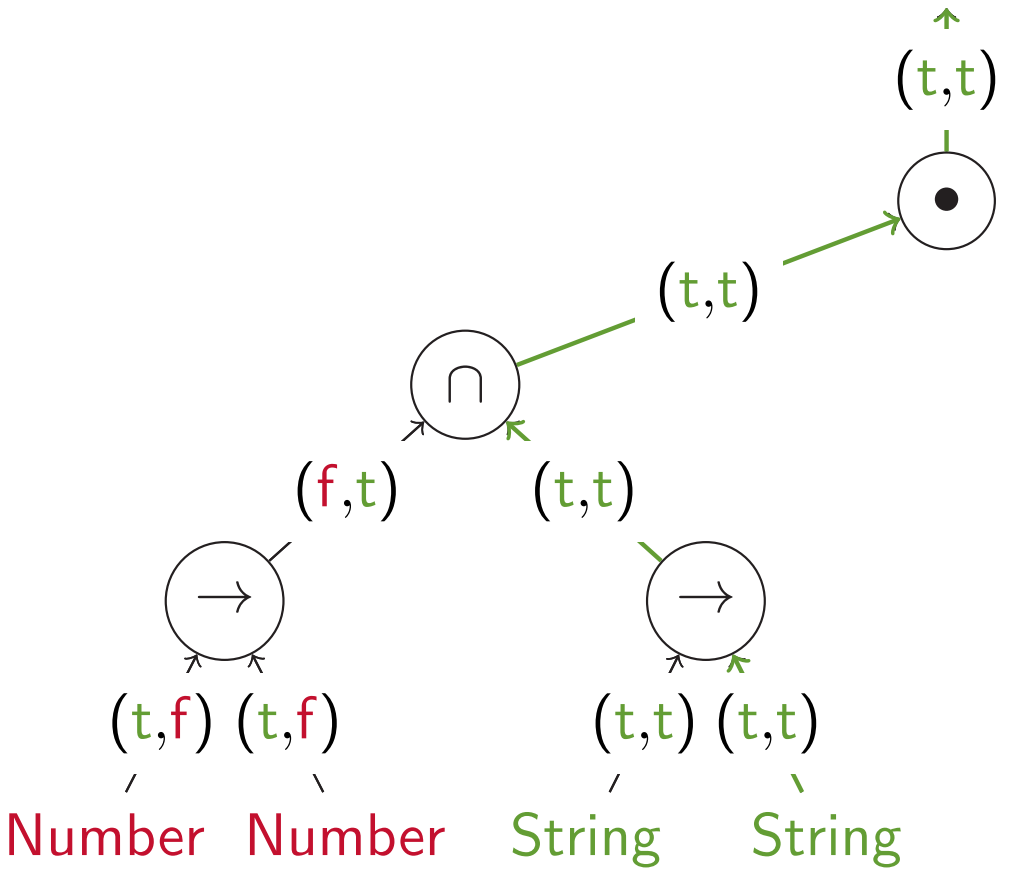
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken('a');*

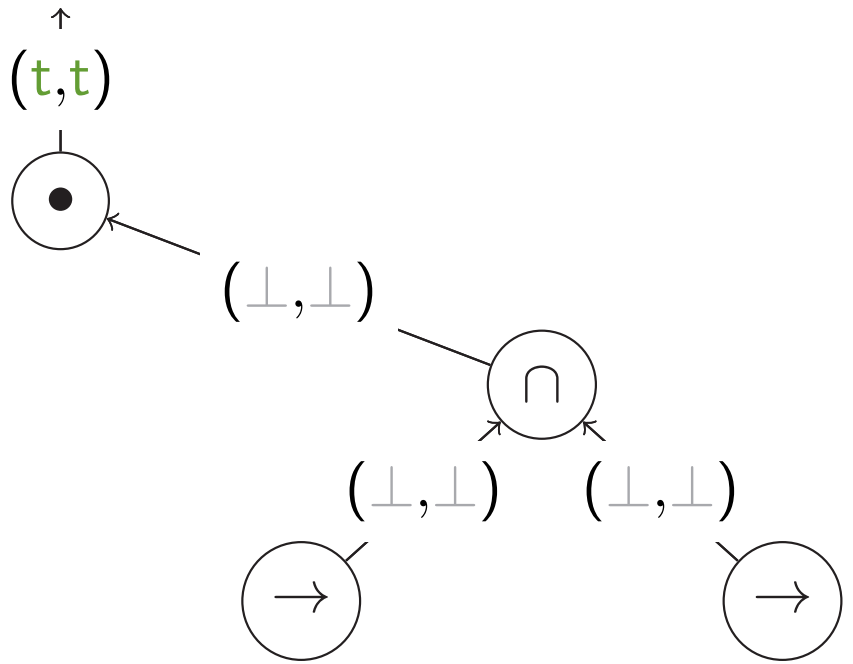
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken('a');* ✓

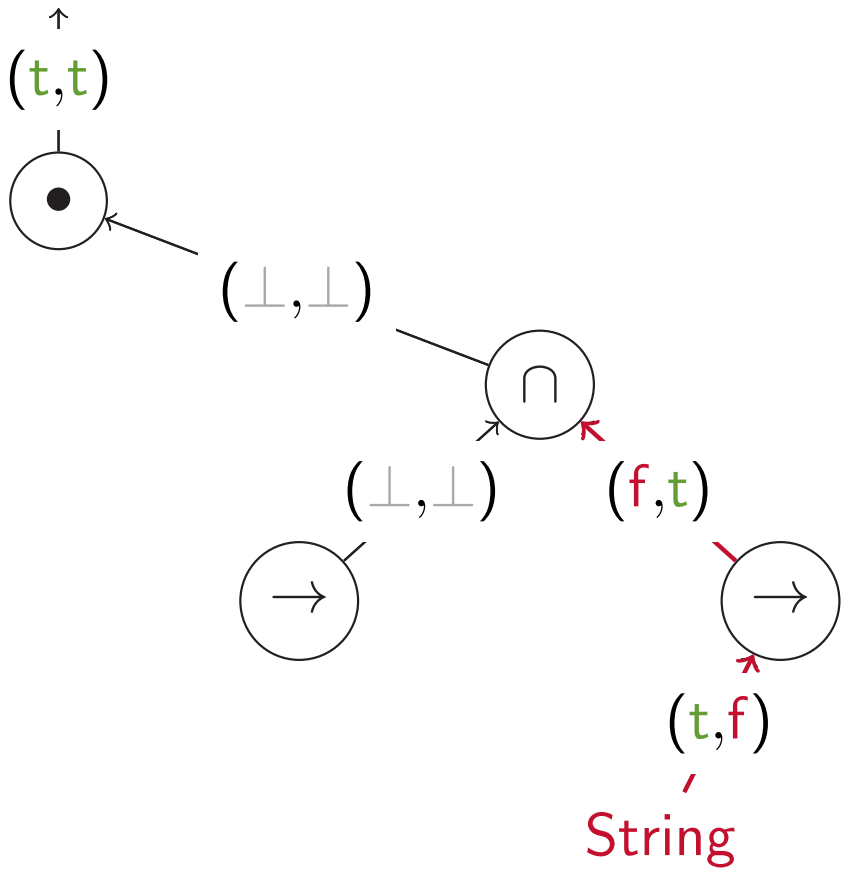
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken(1);*

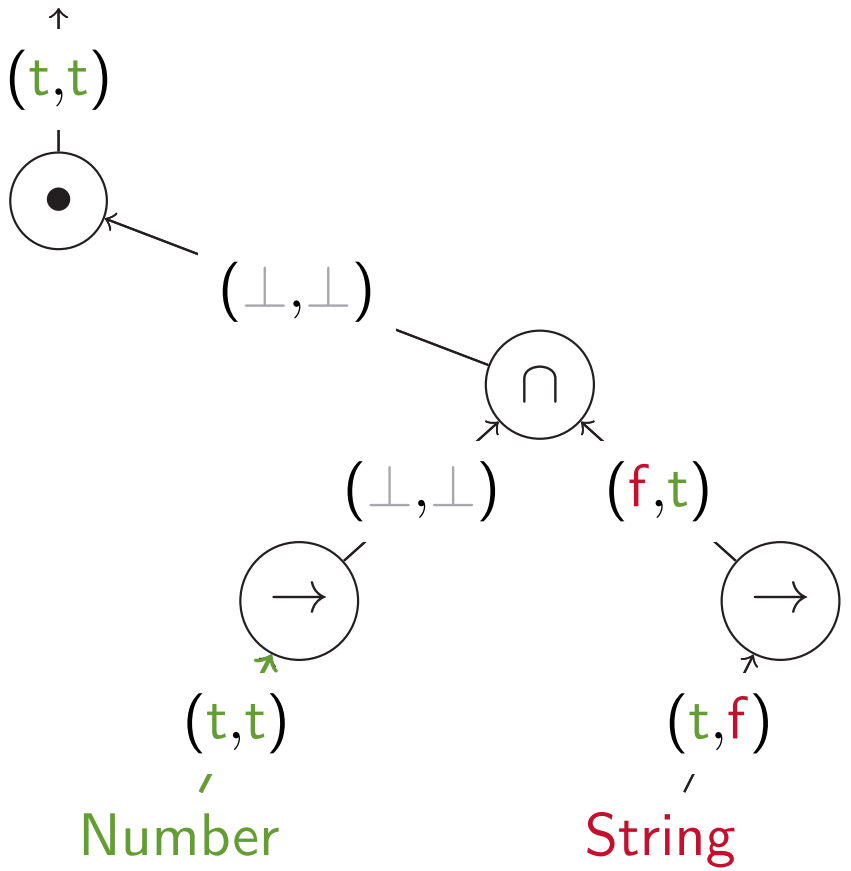
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken(1);*

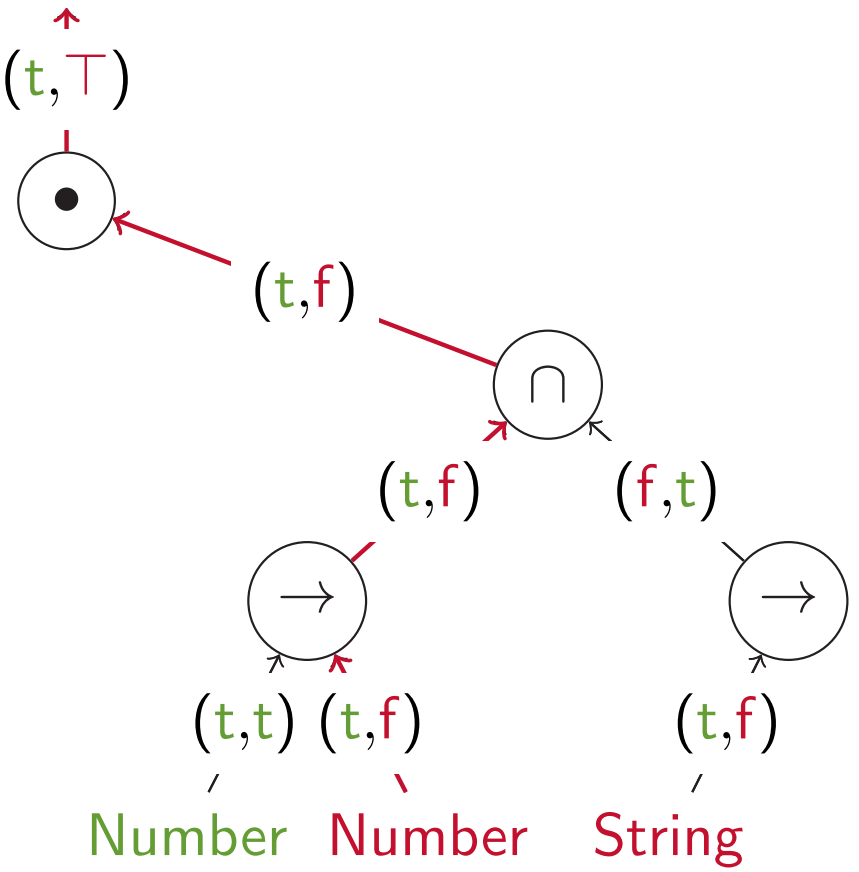
Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken(1);*

Callback Graph



---

1 *(Number → Number) ∩ (String → String)*
2 *addOneBroken(1); blame the subject*

Non-Interference

UNI FREIBURG

- 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);
```

Non-Interference

UNI FREIBURG

- 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_);
```

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  });
```

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_);
```

Contract Constructor

UNI FREIBURG

- 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');
```

Contract Abstraction

UNI FREIBURG

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

Contract Abstraction

UNI FREIBURG

```
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  });
```

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__);
```

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); ✓
```

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  });
```

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); ✓
```

Conclusion

UNI FREIBURG

- 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

Object Contract

- Defined by a mapping from property names to contracts
- Represented as a pair of a *getter* and a *setter* function
- *Getter* and *Setter* apply the property's contract

```
1  var arraySpec = Contract.AObject({length:_Number_});
```

Object Contract

UNI FREIBURG

- Defined by a mapping from property names to contracts
- Represented as a pair of a *getter* and a *setter* function
- *Getter* and *Setter* apply the property's contract

```
1  var arraySpec = Contract.AObject({length:_Number_});
2  var faultyObj = assert({length:'1'}, arraySpec);
```

Object Contract

- Defined by a mapping from property names to contracts
- Represented as a pair of a *getter* and a *setter* function
- *Getter* and *Setter* apply the property's contract

```
1  var arraySpec = Contract.AObject({length:_Number_});
2  var faultyObj = assert({length:'1'}, arraySpec);
3  var faultyObj.length; ✗ blame the subject
```

Object Contract

UNI FREIBURG

- Defined by a mapping from property names to contracts
- Represented as a pair of a *getter* and a *setter* function
- *Getter* and *Setter* apply the property's contract

```
1  var arraySpec = Contract.AObject({length:_Number_});
2  var faultyObj = assert({length:'1'}, arraySpec);
3  var faultyObj.length = '2'; ✗ blame the context
```

Function Contract

- *Function Contract* is build from one contract for the domain and one contract for the range of a function
- An *Object Contract* serves as the domain portion of a *Function Contract* with zero or more arguments
- **AFunction** defines an *Object Contract* from the array

```
1 Contract.AFunction([_Number_, _Number_], _Number_));
```

Function Contract

UNI FREIBURG

- *Function Contract* is build from one contract for the domain and one contract for the range of a function
- An *Object Contract* serves as the domain portion of a *Function Contract* with zero or more arguments
- **AFunction** defines an *Object Contract* from the array

```
4 Contract.AFunction([_Number_, _Number_], _Number_));

5 Contract.Function(
6   Contract.AObject([_Number_, _Number_]), _Number_));
```

Recursive Contract

- Similar to *Constructors*
- Unrolls the constructor when asserted
- Contract is given as argument to the constructor

```
1  var __LinkedList__ = Contract.Recursive(
2    Contract.Constructor(function(__LinkedList__) {
3      return Contract.AObject({
4        value:_Number_,
5        next:__LinkedList__});
6  }));
```

Scores

UNI FREIBURG

Benchmark	Full	System	Baseline
Richards	0.391	0.519	11142
DeltaBlue	0.276	0.360	17462
Crypto	11888	12010	11879
RayTrace	1.09	1.45	23896
EarleyBoyer	5135	5292	5370
RegExp	1208	1181	1207
Splay	20.6	27.8	9555
SplayLatency	73.1	99.7	6289
NavierStokes	6234	7159	12612
pdf.js	9191	9257	9236
Mandreel	12555	12542	12580
MandreelLatency	18741	18883	19398
Gameboy Emulator	6.80	9.07	23801
Code loading	6245	6785	9324
Box2DWeb	3.57	4.67	12528
zlib	29108	28708	29185
TypeScript	187	248	11958

Scores

UNI FREIBURG

Benchmark	Full	w/o C	w/o D	w/o M	w/o P
Richards	0.391	0.582	0.782	0.781	0.903
DeltaBlue	0.276	0.409	0.544	0.544	0.625
Crypto	11888	11912	11914	11986	11979
RayTrace	1.09	1.82	2.51	2.51	3.02
EarleyBoyer	5135	5126	5205	5233	5242
RegExp	1208	1205	1199	1212	1178
Splay	20.6	31.2	42.5	42.5	49.7
SplayLatency	73.1	109	151	151	177
NavierStokes	6234	7924	9176	8943	9456
pdf.js	9191	9548	9156	9222	9152
Mandreel	12555	12586	12549	12346	12431
MandreelLatency	18741	18741	18883	19027	18955
Gameboy Emulator	6.80	10.8	14.9	14.9	17.7
Code loading	6245	6937	7372	7335	7533
Box2DWeb	3.57	5.72	7.80	7.82	9.19
zlib	29108	29025	29047	28926	29063
TypeScript	187	290	400	396	463

Statistic

Benchmark	Contract		
	A	I	P
Richards	24	1599377224	935751200
DeltaBlue	54	2319477672	1340451212
Crypto	1	5	3
RayTrace	42	687240082	509234422
EarleyBoyer	3944	89022	68172
RegExp	0	0	0
Splay	10	11620663	7067593
SplayLatency	10	11620663	7067593
NavierStokes	51	48334	39109
pdf.js	3	15	9
Mandreel	7	57	28
MandreelLatency	7	57	28
Gameboy Emulator	3206	141669753	97487985
Code loading	5600	34800	18400
Box2DWeb	20075	172755100	112664947
zlib	0	0	0
TypeScript	4	12673644	8449090

Statistic

Benchmark	Sandbox M	Sandbox D
Richards	4678756000	4
DeltaBlue	6702256060	5
Crypto	15	3
RayTrace	2546172110	4
EarleyBoyer	340860	6
RegExp	0	0
Splay	35337965	5
SplayLatency	35337965	5
NavierStokes	195545	5
pdf.js	45	4
Mandreel	140	4
MandreelLatency	140	4
Gameboy Emulator	487439925	5
Code loading	92000	4
Box2DWeb	563324735	5
zlib	0	0
TypeScript	42245450	2

Statistic

UNI FREIBURG

Benchmark	Callback
Richards	3351504000
DeltaBlue	4744203248
Crypto	13
RayTrace	2190186074
EarleyBoyer	309120
RegExp	0
Splay	26231845
SplayLatency	26231845
NavierStokes	177197
pdf.js	39
Mandreel	128
MandreelLatency	128
Gameboy Emulator	399084085
Code loading	70400
Box2DWeb	469141435
zlib	0
TypeScript	33796350

Timings

Benchmark	Full	Baseline
Richards	1 day, 18 hours, 21 min, 20 sec	8 sec
DeltaBlue	2 days, 10 hours, 36 min, 49 sec	4 sec
Crypto	9 sec	8 sec
RayTrace	23 hours, 12 min, 37 sec	4 sec
EarleyBoyer	1 min, 9 sec	53 sec
RegExp	9 sec	8 sec
Splay	19 min, 19 sec	3 sec
SplayLatency	19 min, 19 sec	3 sec
NavierStokes	11 sec	4 sec
pdf.js	6 sec	6 sec
Mandreel	5 sec	5 sec
MandreelLatency	5 sec	5 sec
Gameboy Emulator	4 hours, 28 min, 28 sec	5 sec
Code loading	12 sec	9 sec
Box2DWeb	5 hours, 19 min, 49 sec	6 sec
zlib	11 sec	11 sec
TypeScript	22 min, 46 sec	24 sec

Timings

Benchmark	Predicate	Baseline
Richards	1 day, 6 hours, 59 min, 42 sec	8 sec
DeltaBlue	1 day, 20 hours, 58 min, 17 sec	4 sec
Crypto	8 sec	8 sec
RayTrace	17 hours, 1 min, 25 sec	4 sec
EarleyBoyer	1 min, 3 sec	53 sec
RegExp	8 sec	8 sec
Splay	13 min, 55 sec	3 sec
SplayLatency	13 min, 55 sec	3 sec
NavierStokes	9 sec	4 sec
pdf.js	6 sec	6 sec
Mandreel	5 sec	5 sec
MandreelLatency	5 sec	5 sec
Gameboy Emulator	3 hours, 13 min, 13 sec	5 sec
Code loading	12 sec	9 sec
Box2DWeb	3 hours, 52 min, 34 sec	6 sec
zlib	12 sec	11 sec
TypeScript	17 min, 8 sec	24 sec