Typbasierte Analyse von JavaScipt

Phillip Heidegger
heidegger@informatik.uni-freiburg.de

Albert Ludwigs Universität Freiburg

28. Oktober 2008

Gliederung

Motivation

Wieso JavaScript?

- wichtigste Clientsprache im Internet
- wenige Tools vorhanden, die statische Analysen für JavaScript anbieten
- wird professionell eingesetzt

Motivation

Eigenschaften von JavaScript

- dynamisches Typsystem
- automatische Konvertierung von Werten
- Objekte besitzen dynamischer Menge von Eigenschaften
- ...

$\Rightarrow$ Das Verhalten von JavaScript Programmen ist schwer zu sehen.
Noch schwerer ist es, Sicherheit von JavaScript Programmen zu garantieren.

Einleitung in JavaScript

Beispiel:

```
1  var x;
2  x = new Object();
3  x.a = "2";
4  x.b = 84;
5  x.getA = function () { return this.a; };
6  x.mapB = function (f) { return f(this.getB()); };

7  // call method of object x
8  firstcall = x.getA();

9  // create anonymous function as parameter
10 theAnswer = x.mapB ( function (x) { return x/firstcall; } );
```

Einleitung in JavaScript

Beispiel:

1
```
1 var a = "a string";
2 a.x = 51;
3
3 alert(a.x);
```

Einleitung in JavaScript

Beispiel:

1.
```
1 var a = "a string";
2 a.x = 51;
3
3 alert(a.x);
```

2.
```
1 var a = new String("a string");
2 a.x = 51;
3
3 alert(a.x);
```

Typsystem für JavaScript

1. singleton typs für Floats und Strings
2. union typs um Objektzugriff zu verbessern
3. Objekte: Information über Wrapper, Funktionen und Eigenschaften
4. Konvertierungen werden durch Relationen und durch die Wrapperinformationen der Objekte behandelt
5. Funktionen erhalten intersection typs

Typsystem für JavaScript

Probleme

1. ```
   1 eval( e );
   ```

$\longrightarrow$ hybrid typs, gradual typs
   ```

Typsystem für JavaScript

Probleme

1.
```
1 eval( e );
```

→ hybrid typs, gradual typs

2.
```
1 var x = new ... ;
2 x.f = function ... ;
3 x.f ( ) ;
```

→ recency abstraction

Typsysteme für dynamisch getypte Sprachen

Sage: Hybrid checking for flexible specifications *[Gronski u.a (2006)]*

1. Theorem Beweiser um Constraints zu lösen
2. Falls Theorem Beweiser fehlschlägt, dynamischen Check einführen
3. unentscheidbar

Typsysteme für dynamisch getypte Sprachen

Gradual Typing for functional languages *[Siek and Taha (2006)]*
Gradual Typing for Objects *[Siek and Taha (2006)]*

- Kombination aus dynamisch und statisch getypten Programmteilen möglich
- erlaubt Upcasts und Downcast
- Letztere werden dynamisch ueberprueft

Typsysteme für dynamisch getypte Sprachen

Kombination der beiden Ansätze

Well-typed Programs Can't Be Blamed *[P. Wadler, R. Findler (2007)]*

- Beweis, dass dynamisch getypter Teil der Sprache für Typfehler verantwortlich ist
- einfaches Typsystem
- bringt die beiden Ansätze in einen Kontext

Wie ist dies auf JavaScript anwendbar?

Zusammenfassung

Gelöst:

1. Konvertierungen
2. Objekte mit dynamischen Eigenschaften, ohne Zyklen
3. Funktionen mit unterschiedlichen Typen aufrufbar

Probleme:

1. System sehr groß, deshalb Beweise entsprechend schwer überblickbar
2. Interaktion getypter und ungetypter Teile, `eval`
3. Funktionsaufrufe nach Zuweisung