Analyzing JavaScript Programs

Phillip Heidegger
heidegger@informatik.uni-freiburg.de

Albert Ludwigs Universität Freiburg

2. Mai 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);
```

Typesystem

Eigenschaften

1. Singletontypes für Floats und Strings
2. Uniontyps 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 Intersectiontypes

Constraints

$$
\begin{array}{rrl}
 & C & := \texttt{true} \mid \texttt{false} \\
 & & \mid \; C \wedge C \\
\text{Subtyp} & & \mid \; \alpha <: \alpha \mid \tau <: \alpha \mid \alpha <: \tau \\
\text{Subtyp bei Fehlern} & & \mid \; \alpha <:_e \alpha \\
\text{Konvertierung} & & \mid \; \alpha \gg_o \alpha \mid \alpha \gg_s \alpha \mid \alpha \gg_e \alpha \\
\text{Konvertierung} & & \mid \; \alpha \gg_{\text{fun}} \alpha.\, \alpha \to \alpha \\
\text{Zugriff auf Eigenschaften} & & \mid \; \texttt{Read}(\alpha, \alpha, \alpha) \mid \texttt{Write}(\alpha, \alpha, \alpha) \\
\text{Setzen vom Defaulttyp} & & \mid \; \texttt{Default}(\alpha, \alpha) \\
\text{Statement Sequenz} & & \mid \; \texttt{Sequenz}(\alpha, \alpha, \alpha) \\
\text{Rückgabe von Funktionen} & & \mid \; \texttt{NoRetToUndef}(\alpha, \alpha) \\
\texttt{throw } o \text{ in } o \text{ konvertieren} & & \mid \; \texttt{ErrorToObject}(\alpha, \alpha) \\
\text{Fehler entfernen} & & \mid \; \texttt{RemoveError}(\alpha, \alpha)
\end{array}
$$

Typesystem

Probleme – Annotationen

Bei Statements der Form

```
1 a[x] = x;
2 for (a in b) { s; };
3 delete a[x];
```

sind die Annotationen der Objekteigenschaften wichtig. Es gibt:

1. ReadOnly
2. DontEnum
3. DontDelete
4. Internal

Typesystem

Probleme

Zyklen im Objektgraph:

```
1 o.x = o;
```

Typen Cooinduktiv, da bei Zyklen nur unendliche Typen Lösungen ergeben.
$\Rightarrow \mu$ Typen einführen
Ist das syntaktisch in JavaScript sinnvoll machbar?

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. Zyklen im Objektgraph
3. Annotationen der Objekte

Optional – Typen

$\mathfrak{T} \ni$	τ	$::=$	$b \mid \epsilon \mid \rho \mid \omega \mid \tau \vee \tau \mid \top$	Typen
$B \ni$	b	$::=$	$\bot \mid \mathit{prim} \mid F \mid S \mid b \vee b$	Basistypen
	prim	$::=$	$\texttt{null} \mid \texttt{undef} \mid \texttt{false} \mid \texttt{true}$	primäre Typen
	F	$::=$	$f \mid \top_f$	Floattypen
	S	$::=$	$s \mid \top_s$	Stringtypen
	ϵ	$::=$	$\texttt{throw}\ \omega$	Fehlertypen
	ρ	$::=$	$\texttt{noRet}$	Returntyp
	ω	$::=$	$\top_o \mid \{\mathit{wrapper}, \mathit{fun}, \mathit{prop}^*, \tau\}$	Objekttypen
	$\mathit{wrapper}$	$::=$	b	Wrapper
	fun	$::=$	$\bot \mid \tau.\ \omega \rightarrow \tau \mid \mathit{fun} \wedge \mathit{fun}$	Funktionen
	prop	$::=$	$(\mathit{label} : \tau)$	Eigenschaften
	label	$::=$	$s \mid \texttt{this} \mid \text{proto}$	Labels