# Typebasierte Analyse von JavaScript

Phillip Heidegger

Arbeitsbereich Programmiersprachen
Institut für Informatik
Universität Freiburg

Der AJAX Trend sorgt für eine starke Verbreitung von JavaScript und unterstreicht die Bedeutung der Sprache. Um die Zuverlässigkeit der entwickelten Software sicherzustellen und ein professioneller Einsatz von JavaScript zu ermöglichen, werden Analysewerkzeuge benötigt, wie diese für viele andere Programmiersprachen erhältlich sind.

Es gibt viele Ansätze, JavaScript sicherer zu machen. Eine Arbeit von Yu u.a. schlägt z.B. eine dynamische Methode vor [5]. Sie definiert eine Regelmenge, die JavaScript Quellcode dynamisch verändert. Dies ermöglicht dem Besucher einer Webseite, sich eine für ihn angepasste Regelmenge zu erstellen, die festlegt, welche Aktionen der JavaScript Quellcode ausführen darf. Bei einer solchen Lösung wird dem dynamischen Charakter von JavaScript Rechnung getragen und es werden Funktionen wie `eval`, `Document.write` u.a. unterstützt. Allerdings bietet eine solche Lösung für JavaScript Sicherheitsprobleme einem Programmierer keine Hilfestellung, Fehler in seinem Programm zu finden. Der Adressat des von Yu u.a. vorgestellten Ansatzes ist der Benutzer des Internetbrowsers.

Ist aber das Ziel, für einen JavaScript Programmierer Analysemethoden zu entwickeln, die ihm helfen seine Programme vor der Auslieferung an den Kunden auf Fehler zu analysieren, erfordert dies eine statische Analyse des Quellcodes. Als Beispiel sei hier ein Onlineportal eine Internetbank genannt. Dynamische Methoden würden erst nach Auslieferung oder bestenfalls beim Testen zu Ergebnissen führen. Bei einem Internetportal im sicherheitskritischen Umfeld ist dies nicht ausreichend.

Für statische Analysen muss angenommen werden, dass dem Programmierer der vollständige Quellcode seiner Anwendung statisch zur Verfügung steht. Dies bedeutet z.B., dass die Funktion `eval` mit statischen Analysen nicht zu behandeln ist.

Wie alle statischen Methoden, die nicht triviale Eigenschaften über Programmen berechnen, wird eine statische Analysen für JavaScript Näherungen durchführen müssen. Hierbei gibt es zwei Möglichkeiten. Lindahl u. Sagonas stellen in ihrem Artikel einen Ansatz mit Successtypen vor [3]. Hier werden nur Programme zurückgewiesen, bei denen garantiert wird, dass an einer Stelle des Codes ein Fehler besteht. Falls nicht sicher ist, ob der Code korrekt ausgeführt werden kann, wird bei Successtypen das Programm nicht vom Typsystem zurückgewiesen.

Die zweite Möglichkeit besteht in einer konservativen Analyse des Quellcodes, der nur korrekte Programme bzgl. eines Typsystems als gültig einordnet. Für gültige Programme ist eine gewisse Menge Eigenschaften garantiert. So stellt

das Typsystem von Java z.B. sicher, dass bei einer Multiplikation keine Strings mit Zahlen multipliziert werden [1].

In Vortrag wird die zweite Methodik gewählt und eine korrekte Analyse für die Scriptsprache JavaScript kurz vorgestellt. Würde die Eigenschaft, keine Zahlen mit Strings multiplizieren zu können, auf ein Typsystem für JavaScript übertragen, so würde der JavaScript Ausdruck `x = "2" * 4;` vom Typsystem als ungültig abgelehnt. Das Multiplizieren des Strings `"2"` und der Zahl 4 wäre unzulässig. In JavaScript werden Werte aber kontextabhängig in passende Werte anderen Typs konvertiert und der Ausdruck liefert die Zahl 8. Ein Typsystem, das die Multiplikation von Strings und Floats generell ablehnt, würde somit Programme zurückweisen, die ein korrektes Laufzeitverhalten besitzen. Dies wird sich im Allgemeinen nicht komplett vermeiden lassen, wenn man auf Korrektheit Wert legt, aber es muss zumindest sichergestellt werden, dass die einfachen Konvertierungen vom Typsystem behandelt werden können. Der im Vortrag vorgestellte Ansatz basiert stark auf der Arbeit ,,Towards a Type System for Analysing JavaScript Programs“ von Peter Thiemann [4].

Vor der Vorstellung des Typsystems wird JavaScript anhand einiger Beispiele dargestellt. Hierdurch soll ein Gefühl für die Sprache vermittelt werden, und auf einige Fehlerquellen hingewiesen werden, die für einen Programmierer schwer zu erkennen sind.

Ein kurzes Beispiel für einen Fehler in JavaScript Programmen, der sehr schwer zu finden ist, und der auf dem Konvertierungsverhalten von JavaScript basiert, wird hier kurz erwähnt.

```
var y = 0;
y.x = "Hallo";
alert(y.x);
```

Der angegebene Programmausschnitt erstellt eine Variable `y`, und weist dieser den Floatwert `0.0` zu. Durch die automatische Konvertierung wird in der nächsten Zeile einem Objekt, das durch Konvertierung der Zahl in ein Objekt entstanden ist, die Eigenschaft `x` gegeben, und diese auf den Wert `"Hallo"` gesetzt. Leider wird dieser Wert in der dritten Zeile nicht ausgegeben, denn das Objekt wurde nicht an die Variable `y` gebunden. Aus diesem Grund wird in Zeile drei nochmals ein zum Floatwert 0.0 passendes, anderes Objekt erstellt. Dies besitzt keine Eigenschaft `x`. Somit erscheint `undefined` auf dem Bildschirm (Details siehe [4] oder [2]).

An dem Beispiel sieht man eine der zentralen Herausforderungen, denen sich ein Typsystem für JavaScript stellen muss, neben den Konvertierungen von Werten. Objekte von JavaScript haben die Fähigkeit, dynamisch zur Laufzeit Ihre Eigenschaften, sowohl den Wert, als auch die Existenz der Eigenschaften, zu verändern.

Es folgt eine kurze Vorstellung des Typsystems aus meiner Diplomarbeit [2]. Das Typsystem beinhaltet Union- und Intersectiontyps, behandelt die automatischen Konvertierungen von Werten in JavaScript, und kann mit den dynamischen Objekte und der Fehlerbehandlung vom JavaScript umgehen. Es stellt z.B. si-

cher, dass Fehler, wie im Programmausschnitt dargestellt, nicht möglich sind, aber es akzeptiert den Ausdruck `"2" * 4`.

Ein paar Beispiele gültiger JavaScript Programme, die das vorhandene Typsystem noch zurückweist, werden im Vortrag vorgestellt und bildet eine Überleitung zu der Frage, wie das Typsystem geeignet erweitert werden kann, um auch mit den in diesen Beispielen angesprochenen Problemen umgehen zu können. Wie und ob dies möglich ist, ist noch unklar. Es werden ein paar mögliche Ansätze diskutiert. Das vorhandene Typsystem erlaubt es bis jetzt lediglich, Constraints zu generieren. Lösungstrategien dieser Constraints stellt ein weiteres Diskussionsthema dar.

## Literatur


1. James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification, The (3rd Edition) (Java Series)*. Addison-Wesley Professional, July 2005.
2. Phillip Heidegger. Typbasierte Werkzeuge für Fehlersuche und Wartung von JavaScript Programmen, 2007. Deutschland, Universität Freiburg, Diplomarbeit.
3. Tobias Lindahl and Konstantinos Sagonas. Practical type inference based on success typings. In *PPDP '06: Proceedings of the 8th ACM SIGPLAN symposium on Principles and practice of declarative programming*, pages 167–178, New York, NY, USA, 2006. ACM Press.
4. Peter Thiemann. Towards a type system for analyzing javascript programs. In *European Symposium On Programming*, pages S. 408 – 422, 2005.
5. Dachuan Yu, Ajay Chander, Nayeem Islam, and Igor Serikov. Javascript instrumentation for browser security. 2007.