Statisches Typen von JavaScript Programmen

Phillip Heidegger, Peter Thiemann
{heidegger,thiemann}@informatik.uni-freiburg.de

Albert Ludwigs Universität Freiburg

30. April 2008

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

Funktionsaufruf nach Zuweisung

Beispiel

```
1  var x = new Object();    // x ist leeres Objekt
2  var u = x.f;             // u == undefined
3  x.f = function ... ;
4  x.f ( ) ;
```

- `x.f` : `undefined` $\vee$ $(\tau \to \tau')$
- So ist Funktionsaufruf in Zeile 4 nicht möglich!
- Typeänderung von `x.f` löst das Problem

Problem:
Typeänderung von `x.f` wirkt sich auf alle `x`-Objekte aus. Dies wäre unsound.

Funktionsaufruf nach Zuweisung

Typen veränderbar und System sound?

- Führe Abbildungen ein, die über die Typen der Objekte abstrahieren
  - separate Behandlung des neusten `x`–Objekts
  - und aller anderen alten `x`-Objekte
- Typsystem ist kontextsensitiv, und kann für das neuste `x`–Objekt die Typen verändern

Syntax
Labels

→ Dies erfordert das taggen der **new**–Statements mit Labels

```
1  var x = newˆℓ Object(); // Label ℓ
2  var u = x.f;            // u == undefined
3  x.f = function ... ;
4  x.f () ;
```

- Mit ,,den x–Objekten“ sind diejenigen Objekte gemeint, die mit dem **new**–Statement, dass mit Label ℓ getagged ist, markiert sind.
- Die rechte Seite in der 1. Zeile gibt eine Referenz zurück, die mit dem Label ℓ markiert ist

→ Referenzen können anhand dieser Tags gruppiert werden

Anpassung der Semantik
Referenzen

Teile nun die Referenzen in zwei Kategorien ein, damit das neuste Objekt von den anderen unterschieden werden kann:

- Jede Referenzen bekommt die Form $(q\ell, i)$, wobei $q \in \{@, \circ\}$ und $i \in \mathbb{N}$
- $@$ steht für die neueste Referenz
- $\circ$ steht für die alten Referenzen

Anpassung der Semantik

Heaps

Aufteilung des Heaps in zwei Bereiche (H, H_n), den mit neuen und den mit alten Referenzen.

$$H, H_n, e \longrightarrow H', H'_n, e'$$

Zwei Regeln, Lesen von Eigenschaften:

$$H, H_n, (@\ell, i).a \longrightarrow H, H_n, H_n(\ell, i)(a)$$
$$H, H_n, (^\circ\ell, i).a \longrightarrow H, H_n, H(\ell, i)(a)$$

Anpassung der Semantik
Eigenschaften der Heaps

- Heaps haben als Domain Label $\times \mathbb{N}$ und liefern Werte.
- pro @ℓ gibt es nur einen Eintrag in H_n
- Referenzen im ungenauen Heap zeigen immer nur auf den ungenauen Heap
- Referenzen im genauen Heap können auf den genauen oder ungenauen Heap zeigen

$\longrightarrow$ präzise Referenzen müssen an den richtigen Stellen umgewandelt werden und unpräzise werden

$\longrightarrow$ Objekte müssen an der richtigen Stellen vom präzisen Heap in den globalen Heap verschoben werden

Semantik
Beispiel

Sei $H = []$ und $H_n = []$.

```
1  var x = new^ℓ Object(); // Label ℓ
2  var u = x.f;            // u == undefined
3  x.f = function ... ;
4  x.f();
```

- Nach der 1. Zeile:

$$[x \mapsto (@\ell, 0)], [], [(\ell, 0) \mapsto \lambda s.\texttt{undefined}]$$

- Nach der 3. Zeile

$$[x \mapsto (@\ell, 0)], [], [(\ell, 0) \mapsto \lambda s.\texttt{undefined}[f \mapsto \ldots]]$$

Semantik
Beispiel

Sei $H = [(\ell, 0) \mapsto \ldots]$ und $H_n = []$ vor dem Aufruf der Funktion.

```
1 function g() {
2   var x = new^ℓ Object(); // Label ℓ
3   var u = x.f;            // u == undefined
4   x.f = function ... ;
5   x.f();
6   return x;
7 }
```

- Nach der 2. Zeile:

$$[x \mapsto (@\ell, 1)], H, [(\ell, 1) \mapsto \lambda s.\texttt{undefined}]$$

- Nach der 4. Zeile

$$[x \mapsto (@\ell, 1)], H, [(\ell, 1) \mapsto \lambda s.\texttt{undefined}[f \mapsto \ldots]]$$

$\rightarrow$ Transfer vom Objekt (ℓ, i), $\forall i \in \mathbb{N}$ entweder bei Aufruf von g, oder beim **return** Statement innerhalb der Funktion.
Sonst würde in H_n zwei Objekte liegen, die an der Stelle ℓ erzeugt wurden.

Typsytem
Typumgebungen

- Ω und Σ typen die beiden Heaps
- Sie ,,vergessen“ dabei die zweiten Teile (i) der Referenzen, d.h.

$$\Omega : \text{Label} \to \tau \qquad\qquad \Sigma : \text{Label} \to \tau$$

- Objekte bekommen Typen der Form:

$$\texttt{obj}(^\circ\ell) \qquad\qquad \texttt{obj}(@\ell)$$

Typsytem

Typen des kleinen Beispiels

```
1  var x = new^ℓ Object(); // Label: l        =: s1
2  var u = x.f;                               =: s2
3  x.f = function ... ;                       =: s3
4  x.f();
```

1. Zeile:

$$[x \mapsto \texttt{obj}(@\ell), u \mapsto \texttt{undefined}], \Omega, \Sigma \vdash s_1 \\ \rightarrow \Sigma[\ell \mapsto \lambda s.\texttt{undefined}] =: \Sigma_2$$

2. Zeile:

$$[x \mapsto \texttt{obj}(@\ell), u \mapsto \texttt{undefined}], \Omega, \Sigma_2 \vdash s_2 \rightarrow \Sigma_2$$

3. Zeile:

$$[x \mapsto \texttt{obj}(@\ell), u \mapsto \texttt{undefined}], \Omega, \Sigma_2 \vdash s_3 \\ \rightarrow \Sigma[\ell \mapsto \{f \mapsto \tau_f, \lambda s.\texttt{undefined}\}] =: \Sigma_3$$

4. Zeile typbar, da an der Stelle $@\ell$ `f` nur einen Funktionstypen besitzt.

Typsystem
Beispiel mit Funktion

Sei $H = [(\ell, 0) \mapsto \ldots]$ und $H_n = []$ for Aufruf der Funktion.

```
1 function g() {
2   var x = new^ℓ Object(); // Label ℓ
3   var u = x.f;            // u == undefined
4   x.f = function ... ;
5   x.f();
6   return x;
7 }
```

- Ω jetzt interessant, den es ist zuständig für alle inexakten Referenzen, d.h. alle Objekte aus H.
  - $\Omega = [\ell \mapsto \{f \mapsto \tau_f, \lambda s.\texttt{undefined}\}]$
- Beim Transfer von einem Objekt aus H_n ins H müssen die Typen von Ω und Σ für das ℓ passen, z.B:

$$\Omega(\ell) = \Sigma(\ell)$$

Zusammenfassung

- Initialisierungen von Objekten √
- Allgemeiner Fall des Transfers von H_n nach H erst beim Funktionsaufruf √
  - Setze ein Effektsystem ein, um Labelmengen zu behandeln
  - Eine Labelmenge für eine Funktion gibt an, welche präzisen Referenzen eine Funktion manipuliert
- Beweis für Soundness √
- Inferenz von Ω, Σ, den Labelmengen, usw. ?
- Äquivalenz der Inferenz mit dem logischen Typesystem ?
- Anwendung auf andere Bereiche von JavaScript ??