Statisches Typen von JavaScript Programmen

Phillip Heidegger and Peter Thiemann

Institut für Informatik, Universität Freiburg
{heidegger,thiemann}@informatik.uni-freiburg.de

1 Syntax

In Abbildung 1 wird eine Syntax in ANF Form angegeben. Hierbei handelt es sich im wesentlichen um eine Erweiterung des Lambda Kalkühls um Ojekte, wie Sie in JavaScript zu finden sind. Objekte werden als Abbildungen von Strings nach Werten angesehen und der Zugriff auf Eingenschaften von Objekten wird wie in JavaScript durch die Notation $v.a$ dargestellt. Durch die Notation $v.a$ wird die Eigenschaft „$\mathtt{a}$“ des Objekts $\mathtt{v}$ gelesen, durch $v_1.a \stackrel{L}{=} v_2$ wird die Eigenschaft „$\mathtt{a}$“ des Objekts $\mathtt{v}_1$ gesetzt. Bei L handelt es sich um eine Labelmenge, deren Bedeutung im Rahmen der statischen Semantik erklärkt wird, und die vorerst ignoriert werden kann.

Im Lambda Ausdruck $\lambda(y,z).e$ entspricht das y dem `this` aus JavaScript und bei dem x handelt es sich um den Funktionsparameter. Der Methodenaufruf bindet somit an das y das entsprechende Objekt. Falls die Funktion nicht als Methode aufgerufen wird, wird das y auf `undefined` gesetzt. Dies ist eine Abweichung zur dynamische Semantik von JavaScript (siehe ECMA), die hier der Einfachheit halber nicht weiter betrachtet werden soll. Die Anpassung der Semantik an die Spezifikation stellt aber kein großes Hinderniss dar (SIEHE Literatur) und verhält sich orthogonal dem hier vorgestellten Ansatz.

$$
\begin{array}{rl}
\mathsf{Expression} \ni e & ::= s \mid \mathtt{let}^L\ x = e\ \mathtt{in}\ e \\
\mathsf{Value} \ni v & ::= x \mid \lambda(y,z).e \mid \mathtt{undefined} \mid \boxed{(q\ell, i)} \\
\mathsf{SeriousExpr} \ni s & ::= v \mid v(v)^L_L \mid \mathtt{new}^\ell \mid v.a \mid v.a \stackrel{L}{=} v \mid v.a(v)^L_L \\
\mathsf{Location} \ni \ell & ::= l_1 \mid l_2 \mid \ldots \\
\mathsf{Location} \supseteq L & \\
\mathsf{Label} \ni a & \\
\mathsf{Type} \ni t & ::= \mathtt{undefined} \mid (\Sigma, t \times t) \stackrel{L}{\to} (\Sigma, t) \mid \mathtt{obj}(q\ell) \mid \top \\
\mathsf{Qualifier} \ni q & ::= {}^\circ \mid @ \\
\mathsf{HeapType} \ni r & \in \mathsf{Label} \xrightarrow{fin} \mathsf{Type} \\
\mathsf{GlobalEnv} \ni \Omega & ::= \emptyset \mid \Omega(\ell : r) \\
\mathsf{LocalEnv} \ni \Sigma & ::= \emptyset \mid \Sigma(\ell : r) \\
\mathsf{TypeEnv} \ni \Gamma & ::= \emptyset \mid \Gamma(x : t)
\end{array}
$$

The gray marked value is only created during evaluation.

Fig. 1: Syntax.

2 dynamische Semantik

In Abbildung 2 finden Sie die dynamische Semantik der in Abbildung 1 vorgestellten Sprache. Der Hauptunterschied zur Semantik des Lambdakalkühls besteht in der Modellierung der Referenzen und des Heaps. Objekte, die durch den selben `new`-Ausdruck erzeugt werden, sollen alle gruppiert werden, und im wesentlich in der statischen Semantik den selben Typ erhalten. Zu diesem Zweck erhalten alle Referenzen auch schon in der dynamischen Semantik das Label des `new`-Ausdrucks. Um nun aber im der statischen Semantik das neuste Objekt zum jeweiligen `new`-Ausdruck gesondert behandelnt zu können, wird in der dynamischen Semantik jede Referenz zustätzlich mit einem Falg markiert (das $q \in \{@, \circ\}$). Hierbei stellt die dynamische Semnatik sicher, dass für jedes Label nur ein einzige Objekt im präzisen Heap vorhanden ist. Dies wird später bei der statischen Betrachtung eine Änderung des Types aller neusten Objekte ermöglichen.

Präzise Referenzen verweisen immer auf den Heap der neuen Objekte (H_0), alle anderen unpräzisen Referenzen beziehen sich auf die alten Versione und verweisen auf den alten Heap (H). Zugriffe auf Objekteigenschaften entsprechen dann dem Dereferenzieren in dem passenden Heap. Dies können Sie z.B. in der Regel zum Auslesen von Eigenschaften aus Objekten sehen (SRead′, SRead). Die präzisen Referenzen werden in dem Heap H_0 nachgeschlagen, die unpräzisen in H.

Eine wichtige Operation in der dynmischen Semantik stellt die Funktion $e^{\natural L}$ dar. Um sicherzustellen, dass für jedes $\ell \in \mathsf{Location}$ nur maximal ein Objekt im präzisen Heap vorhanden ist, müssen an passenden Stellen alle älteren Objekte mit Label l vom präzisen in den unpräzisen Heap verschoben werden. Diese Verschiebung findet sich z.B. in der Regel SApp′. Leider müssen bei jeder Verschiebung alle Referenzen, die sich im Heap oder in den Ausdrücken als Werte befinden, und die auf das entsprechende Objekt zeigten, entsprechend angepasst werden. Ansonsten würden sich die Referenzen auf den falschen Heap beziehen und die Semantik würde unerwartet Speicherzugrifffehler liefern. Die Operation $e^{\natural L}$ verändert alle Referenzen $(@\ell, i)$ mit beliebigen i für alle $\ell \in L$, die in dem Ausdruck e vorkommen, in unpräzise Versionen.

Eine weitere Eigenschaft der dynamischen Semantik besteht darin, dass Objekte des unpräzisen Heaps keine Referenzen auf neue Objekte enthalten können. Dies wird durch die Semantik ebenfalls sichergestellt. Falls sich somit zykliche Objektstrukturen ergeben muss entweder die gesammte zyklische Struktur innerhalb des neuen Heaps liegen, oder aber Sie wird komplett in den Heap der unpräzisen Referenzen verschoben. Die statische Semantik wird nach einer Verschiebung in den unpräzisen Heap keine Typänderungen mehr zulasssen.

$$
\begin{array}{lll}
\mathsf{HeapContent} & = & (\mathsf{Label} \xrightarrow{fin} \mathsf{Value}) \\
\mathsf{Heap} & = & (\mathsf{Location} \times \mathbf{N}) \to \mathsf{HeapContent} \\
H, H_0 & \in & \mathsf{Heap} \\
h & \in & \mathsf{HeapContent} \\
w & ::= & v \mid v.a
\end{array}
$$

$$
\begin{array}{lll}
\textsc{SApp} & H, H_0, (\lambda(y,x).e)(v)^L_L & \longrightarrow H, H_0, e\{y, x \mapsto \texttt{undefined}, v\} \\
\textsc{SLet} & H, H_0, \texttt{let}^L\ x = v\ \texttt{in}\ e & \longrightarrow H, H_0, e\{x \mapsto v\} \\
\textsc{SApp}' & H, H_0, w(v)^L_\emptyset & \longrightarrow H'', H_0'', w(v^{\natural_L})^L_L \\
 & & \quad \text{if } H_L = H_0^{\natural_L} \downarrow \{(\ell, i) \mid i \in \mathbb{N}, \ell \in L\}, H_L \neq \emptyset \\
 & & \quad H'' = H \cup H_L, H_0'' = H_0^{\natural_L} \setminus H_L \\
\textsc{SNew} & H, H_0, \texttt{new}^\ell & \longrightarrow H, H_0 + [(\ell, i) \mapsto \lambda a.\texttt{undefined}], (@\ell, i) \\
 & & \quad \text{if } H_\ell = H_0^{\natural_\ell} \downarrow \{(\ell, i) \mid i \in \mathbb{N}\} \\
 & & \quad H'' = H \cup H_\ell, H_0'' = H_0^{\natural_\ell} \setminus H_\ell, \\
\textsc{SRead} & H, H_0, (@l, i).a & \longrightarrow H, H_0, H_0(l,i)(a) \quad \text{if } (l,i) \in \mathrm{dom}(H_0) \\
\textsc{SRead}' & H, H_0, (^\circ l, i).a & \longrightarrow H, H_0, H(l,i)(a) \quad \text{if } (l,i) \in \mathrm{dom}(H) \\
\textsc{SWrite} & H, H_0, (@\ell, i).a \stackrel{L}{=} v & \longrightarrow H, H_0[(\ell, i) \mapsto H_0(\ell, i)\{a \mapsto v\}], \texttt{undefined} \\
\textsc{SWrite}' & H, H_0, (^\circ\ell, i).a \stackrel{L}{=} v & \longrightarrow H'[(\ell, i) \mapsto H'(\ell, i)\{a \mapsto v^\natural\}], H_0', \texttt{undefined} \\
 & & \quad \text{if } H_L = H_0^{\natural_L} \downarrow \{(\ell, i) \mid \ell \in L, i \in \mathbb{N}\} \\
 & & \quad H' = H \cup H_L, H_0' = H_0^{\natural_L} \setminus H_L \\
\textsc{SMCall} & H, H_0, (@\ell, i).a(v)^L_L & \longrightarrow H, H_0, e\{y, x \mapsto (@\ell, i), v)\} \\
 & & \quad \text{if } H_0(\ell, i)(a) = \lambda(y,x).e \\
\textsc{SMCall}' & H, H_0, (^\circ\ell, i).a(v)^L_L & \longrightarrow H, H_0, e\{y, x \mapsto (^\circ\ell, i), v)\} \\
 & & \quad \text{if } H(\ell, i)(a) = \lambda(y,x).e
\end{array}
$$

$$
\textsc{SLet}' \quad \frac{H, H_0, e \longrightarrow H', H_0', e'}{H, H_0, \texttt{let}^L\ x = e\ \texttt{in}\ e'' \longrightarrow H', H_0', \texttt{let}^L\ x = e'\ \texttt{in}\ e''}
$$

Fig. 2: Small-step operational semantics.

3 statische Semantik

Die statische Semantik des Systems finden Sie in Abbildung reffig:typing-rules. Jeder Wert wird durch ein Typurteil der Form

$$\Omega, \Sigma, \Gamma, \vdash_v v : \tau \tag{1}$$

beurteilt. Bei Ω handelt es sich um einen Heaptyp, der in Abbildung 1 eingeführt wird, und der für den ungenauen Heap zuständig ist. Dieser globale Heaptyp ist bei allen Typregeln gleich und somit für das gesammte Programm und alle Objekte mit passend gelabelten Referenzen zuständig, wenn man von den neusten Objekten absieht. Diese werden durch den Heaptyp Σ behandelt. Hier ist es möglich, dass sich das Σ von Ausdruck zu Ausdruck verändert und somit eine Inizialisierung von Objekten möglich wird. Das Γ bindet Variablen an Typen. Die Typregeln weisen somit unter den gegebenen drei Umgebungen den Werten einen Typ zu. Werte machen von der Möglichkeit, die lokale Heapumgebung zu verändern keinen Gebrauch.

Diese Möglichkeit wird dann durch die Typeregeln für Ausdrücke

$$\Omega, \Sigma, \Gamma \vdash_e e : \tau \Rightarrow L, \Omega, \Gamma' \tag{2}$$

wahrgenommen. Zusätzlich zu dem Typ liefern die Regeln nach dem $\Rightarrow$ eine Menge von veränderten Labeln L, eine neue veränderte Heapumgebung und eine neue Typumgebung zurück.

In der statischen Semantik muss der Tatsache, dass Objekte an bestimmten Stellen von der dynamischen Semantik vom neuen Heap in der Heap der unpräzisen Objekte verschoben werden, Rechnung getragen werden. Ansonsten wäre das System nicht sound und keinerlei sinnvolle Aussagen von Ihm ableitbar. Die Operationen, die dieser Tatsache gerecht werden finden sich an vielen Stellen der statischen Semantik, z.B. in Form des $\Gamma'' = (\Gamma')^{\natural L'}$ und des $\Omega, \Sigma_2 \vdash_\Gamma \Gamma' \lhd \Gamma''$ in der Typregel FUNCTION. Die erste Relation stellt sicher, dass in der Typumgebung Γ' keine Variablen mit präzisen Objektypen vorhanden sind, die auf Label $\ell \in L$ zeigen. Die zweite Relation modelliert den Datenfluss von präzisen in den unpräzisen Heap für alle Referenzen ℓ, die durch lokale Variablen erreichbar sind, für die Heaptypen.

Die genauen Definitionen dieser Relationen präsentieren wir zu diesem Zeitpunkt in diesem erweiterten Abstrakt der Einfachheit halber nicht. Sie sind natürlich für den Soundness Beweis unverzichtbar, aber für das Verständniss des Systems sind sie nicht von Bedeutung.

Typing rules for values:

$$\textsc{Undefined}\quad \Omega, \Sigma, \Gamma \vdash_v \texttt{undefined} : \texttt{undefined} \qquad \textsc{Object}\quad \Omega, \Sigma, \Gamma \vdash_v (q\ell, i) : \texttt{obj}(q\ell)$$

$$\textsc{Variable}\quad \frac{\Gamma(x) = t}{\Omega, \Sigma, \Gamma \vdash_v x : t} \qquad \textsc{Subsumation}\quad \frac{\Omega, \Sigma, \Gamma \vdash_v v : t \qquad \Omega, \Sigma \vdash_t t \lhd t'}{\Omega, \Sigma, \Gamma \vdash_v v : t'}$$

$$\textsc{Function}\quad \frac{\begin{array}{c}\mathrm{dom}(\Sigma_1) = \mathrm{dom}(\Sigma_2) \subseteq \mathrm{dom}(\Omega) \\ L' \cup L'' \subseteq L \qquad \Gamma' = \Gamma \downarrow \mathrm{fv}(\lambda(y,x).e) \qquad L' = @\mathrm{Locs}_\Sigma(\Gamma') \qquad \Gamma'' = (\Gamma')^{\natural L'} \\ \Omega, \Sigma_2 \vdash_\Gamma \Gamma' \lhd \Gamma'' \qquad \Omega, \Sigma_2, \Gamma''(y : t_0)(x : t_2) \vdash_e e : t_1 \Rightarrow L'', \Sigma_1, \Gamma'''(y : t_0')(x : t_2')\end{array}}{\Omega, \Sigma, \Gamma \vdash_v \lambda(y,x).e : (\Sigma_2, t_0 \times t_2) \xrightarrow{L} (\Sigma_1, t_1)}$$

Typing rules for (serious) expressions:

$$\textsc{Value}\quad \frac{\Omega, \Sigma, \Gamma \vdash_v v : t}{\Omega, \Sigma, \Gamma \vdash_e v : t \Rightarrow \emptyset, \Sigma, \Gamma}$$

$$\textsc{Let}\quad \frac{\Omega, \Sigma, \Gamma \vdash_e e_1 : t_1 \Rightarrow L_1, \Sigma_1, \Gamma_1 \qquad \Omega, \Sigma_1, \Gamma_1(x : t_1) \vdash_e e_2 : t_2 \Rightarrow L_2, \Sigma_2, \Gamma_2(x : t_1')}{\Omega, \Sigma, \Gamma \vdash_e \texttt{let}^{L_1}\ x = e_1\ \texttt{in}\ e_2 : t_2 \Rightarrow L_1 \cup L_2, \Sigma_2, \Gamma_2}$$

$$\textsc{Function Call}\quad \frac{\begin{array}{c}\Omega, \Sigma, \Gamma \vdash_v v_1 : (\Sigma_2, t_0 \times t_2) \xrightarrow{L} (\Sigma', t_1) \\ \Omega, \Sigma \vdash_t \texttt{undefined} \lhd t_0 \qquad \Omega, \Sigma, \Gamma \vdash_v v_2 : t_2 \\ \Gamma' = \Gamma^{\natural L} \qquad \Omega, \Sigma \vdash_\Gamma \Gamma \lhd \Gamma' \qquad \Sigma_2 = \Sigma^{\natural L} \qquad \Omega, \Sigma \vdash_L @L \lhd {}^\circ L \qquad L' \subseteq L\end{array}}{\Omega, \Sigma, \Gamma \vdash_e v_1(v_2)^L_{L'} : t_1 \Rightarrow L, \Sigma', \Gamma'}$$

$$\textsc{Methode Call}\quad \frac{\begin{array}{c}\Omega, \Sigma, \Gamma \vdash_v v_1 : \texttt{obj}(p) \\ \Omega, \Sigma \vdash_r p.a : (\Sigma_2, t_0 \times t_2) \xrightarrow{L} (\Sigma', t_1) \qquad \Omega, \Sigma \vdash_t \texttt{obj}(p) \lhd t_0 \\ \Omega, \Sigma, \Gamma \vdash_v v_2 : t_2 \qquad \Sigma' = \Sigma_2^{\natural L} \qquad \Gamma' = \Gamma^{\natural L} \qquad \Omega, \Sigma \vdash_\Gamma \Gamma \lhd \Gamma' \qquad L' \subseteq L\end{array}}{\Omega, \Sigma, \Gamma \vdash_e v_1.a(v_2)^L_{L'} : t_1 \Rightarrow L, \Sigma', \Gamma'}$$

$$\textsc{New}\quad \frac{\ell \in \mathrm{dom}(\Omega) \qquad \Gamma' = \Gamma^{\natural \ell} \qquad \Omega, \Sigma \vdash_\Gamma \Gamma \lhd \Gamma' \qquad \Omega, \Sigma \vdash_\ell @\ell \lhd {}^\circ \ell}{\Omega, \Sigma, \Gamma \vdash_e \texttt{new}^\ell : \texttt{obj}(@\ell) \Rightarrow \{\ell\}, \Sigma(\ell \mapsto \lambda z.\texttt{undefined}), \Gamma'}$$

$$\textsc{Read}\quad \frac{\Omega, \Sigma, \Gamma \vdash_v v : \texttt{obj}(p) \qquad \Omega, \Sigma \vdash_r p.a : t}{\Omega, \Sigma, \Gamma \vdash_e v.a : t \Rightarrow \emptyset, \Sigma, \Gamma}$$

$$\textsc{Write}\quad \frac{\begin{array}{c}\Omega, \Sigma, \Gamma \vdash_v v : \texttt{obj}(ql) \\ \Omega, \Sigma, \Gamma \vdash_v v' : t' \qquad \Omega, \Sigma \vdash_L @L \lhd qL \qquad L' = @\mathrm{Locs}_\Sigma(t') \\ L' \subseteq L \qquad \Omega, \Sigma \vdash_w ql.a = t' \Rightarrow \Sigma' \qquad \Gamma' = \Gamma^{\natural L'} \qquad \Omega, \Sigma \vdash_\Gamma \Gamma \lhd \Gamma'\end{array}}{\Omega, \Sigma, \Gamma \vdash_e v.a \overset{L}{=} v' : \texttt{undefined} \Rightarrow L, \Sigma', \Gamma'}$$

Fig. 3: Typing rules.