# Een natuurlijke semantiek voor prototype oververing en lexicaal bereik

Kelley van Evert & Tim Steenvoorden

27 mei 2012

# Inhoudsopgave

# 1 Inleiding

- Motivatie
- JavaScript
- Lexical scope – vrije/gebeonden variabelen
- Objecten en prototype overerving

# 2 Notatie en terminologie

## 2.1 Functies

In dit werkstuk identificeren we een functie met zijn grafiek, d.w.z. een functie $f : X \to Y$ is gelijk aan de verzameling paren $(x, y) \in X \times Y$ waarvoor geldt: $f(x) = y$.

## 2.2 Partiële functies

Vrijwel alle functies die we in dit werkstuk behandelen zijn partiële functies. Wanneer een partiële functie $f$ niet gedefiniëerd is op een zeker punt $x$, schrijven we $f(x) = \bot$. Wanneer het omgekeerde het geval is, schrijven we $f(x) \neq \bot$.

Voor een willekeurige term $\phi = \ldots f(x) \ldots$, waarbij $f$ een partiële functie is die niet gedefiniëerd is op punt $x$, geldt dat $\phi = \bot$. Op deze manier hoeven we niet iets omslachtigs te schrijven als: "als $f(x) = \bot$, dan $z = \bot$; anders als $f(x) \neq \bot$, dan $z = \phi$".

## 2.3 Tupels

We zullen meermaals in ons werkstuk gebruik maken van willekeurig grote, maar altijd eindige, "lijsten" van elementen uit een zekere verzameling: *tupels*. Deze tupels worden gerepresenteerd door eindige (partiële) functies $t : \mathbb{N} \to X$, als X de verzameling element in kwestie is, waaraan nog een paar extra voorwaarden worden gesteld. De verzameling van alle tupels op een zekere verzameling $X$, genoteerd $X_{\langle\rangle}$, is als volgt gedefiniëerd:

$$X_{\langle\rangle} \stackrel{\text{def}}{=} \left\{ t : \mathbb{N} \to X \mid \exists_{N \in \mathbb{N}} \left[ \forall_{n<N} [t(n) \neq \bot] \wedge \forall_{n \geq N} [t(n) = \bot] \right] \right\}$$

We schrijven $\langle\rangle$, maar ook wel $\varnothing$, voor de lege tupel (deze is natuurlijk hetzelfde voor elke waarden verzameling $X$).

We schrijven $\langle x_0, x_1, \ldots, x_{N-1}\rangle$ voor de tupel $t$ waarvoor geldt: $\forall_{n<N}[t(n) = x_n]$, en: $\forall_{n\geq N}[t(n) = \bot]$.

Als $t$ een tupel is $\in X_{\langle\rangle}$, en $x$ een element van $X$, dan schrijven we $t : x$ voor de tupel $t' = t[N \mapsto x]$, waarbij $N = \min\{n \in \mathbb{N} \mid t(n) = \bot\}$.

## 2.4 Beschouwing semantisch model

We definieren in dit werkstuk een natuurlijke semantiek, d.w.z. een ?-ste orde logica, met axioma's en deductieregels, en een bijbehorende structuur waarin deze zich afspeelt.

Deze structuur, die we ook wel het *semantisch model* zullen noemen, heeft onderstaand opgesomde elementen. Deze worden verderop precies gedefinieerd, onderstaande opsomming geeft slechts een algemeen beeld.

$\mathbb{M}$
: De verzameling mogelijke *geheugens*, welke ook wel als *eindtoestanden* worden geinterpreteerd.

$(Stm \times \mathbb{M} \times \mathbb{L} \times \mathbb{L})$
: De verzameling *toestanden*, ook wel *configuraties*.

$(\longrightarrow)$
: Een tweeplaatsig predikaat welke als eerste argument een element uit de verzameling van toestanden neemt, en als tweede argument een element uit de verzameling van eindtoestanden ($\mathbb{M}\ldots$). De uitspraak $(S, m, \sigma, \tau) \longrightarrow m'$ moet worden geinterpreteerd worden als:

  > "Het programma $S$, met geheugen $m$, in scope $\sigma$ en met als **this** object $\tau$, resulteert in eindtoestand $m'$, mits $S$ *correct* is".

## 2.5 Notationele conventies

Terwille van elegantie houden we een aantal gebruikelijke notationele conventies aan:

1. Voor elke twee willekeurige tweestemmige predikaten $\mathsf{S}$ en $\mathsf{T}$ (mogelijk ook $=$), en drie willekeurige elementen $a$, $b$ en $c$, definieren we de afkorting:

   $$a \,\mathsf{S}\, b \,\mathsf{T}\, c \stackrel{\text{def}}{=} a \,\mathsf{S}\, b \wedge b \,\mathsf{T}\, c$$

   in het geval dat deze bewering correct getypeerd is.

2. Op eenzelfde manier definieren we ook de volgende afkorting:

   $$\{a \in A \mid \phi\} \stackrel{\text{def}}{=} \{a \mid a \in A \mid \phi\}$$

[...]

# 3 Taal en syntax

In dit hoofdstuk zullen we de taal presenteren waarvoor we een natuurlijk taal construeren. De taal maakt gebruikt van prototype overerving en lexicaal bereik. Eerst zullen we een aantal voorbeeldprogramma's beschouwen, om zo informeel het karakter van de te formaliseren taal over te brengen. Daarna geven we een rigoreuze definitie met behulp van een BNF grammatica. De structuur van de productieregels van grammatica worden in latere hoofdstukken gebruikt om axioma's en deductieregels op te stellen. Daarmee heeft de grammatica in zekere zin een dubbele functie.

Elk voorbeeldprogramma en zijn toelichtingen worden als volg gepresenteerd:

| | | |
|---|---|---|
| A.1 | **local** $f$ | *- variabelen moeten worden gedeclareerd* |
| A.2 | $f =$ **function** $(i)$ **returns** $n$ | |
| A.3 | **local** $n$ | |
| A.4 | $n = 2 \times (i + 5)$ | |
| A.5 | | *- x bestaat niet in deze scope* |
| A.6 | **local** $x$ | *- x is ongedefinieerd (maar wel aanwezig)* |
| A.7 | $x = f(42)$ | *- x = 89* |

De toelichtingen moeten als informeel commentaar worden beschouwd, waarmee we aan proberen te geven hoe het programma zich gedraagt. Vaak zijn het uitspraken over de toestand waarin het programma zich bevindt, direct na de linker regel te hebben "uitgevoerd".

## 3.1 Voorbeeldprogramma's

Een variabele moet gedeclareerd worden, en pas daarna kan er een waarde aan worden toegekend.

| | | |
|---|---|---|
| B.1 | | *- x bestaat niet (in deze scope)* |
| B.2 | **local** $x$ | *- x is ongedefinieerd (maar wel aanwezig)* |
| B.3 | $x = 5$ | *- x = 5* |

Het concept van declaratie is juist in deze taal, gezien het lexicaal bereik van variabelen, heel belangrijk. Vergelijk het bovenstaande programma fragment bijvoorbeeld met de volgende situatie.

Variabelen hebben geen vaste type. Er zijn drie typen waarden in de taal: getallen, functies en objecten.

| | |
|---|---|
| C.1 **local** $x$ | |
| C.2 $x = 5$ | *- de waarde van x is een getal* |
| C.3 $x =$ **function** () { **skip** } | *- de waarde van x is een functie* |
| C.4 $x$ **object** | *- de waarde van x is een object* |

De taal is object georienteerd.

| | |
|---|---|
| D.1 **local** $o$ | |
| D.2 $o$ **object** | |
| D.3 | *- o.f is niet gedefinieerd* |
| D.4 $o.f =$ **function** () { **skip** } | *- toekenning waarde aan object attribuut* |
| D.5 | *- o.f is wel gedefinieerd* |
| D.6 $o.n = 5$ | |

Van de drie typen, zijn getallen en functies *primitief*, en objecten *niet primitief*. Primitieve waarde worden zelf gekopieerd (*by-value*), maar van niet-primitieve waarden worden *referenties* gekopieerd (*by-reference*).

| | |
|---|---|
| E.1 **local** $x$; $x = 6$ | |
| E.2 **local** $y$; $y = x$ | *- x = 6 en y = 6* |
| E.3 $y = 7$ | *- x = 6 en y = 7* |
| E.4 | |
| E.5 **local** $p$; $p.n = 6$ | |
| E.6 **local** $q$; $q = p$ | *- p en q verwijzen nu naar hetzelfde object* |
| E.7 | *- p.n = 6 en q.n = 6* |
| E.8 $q.n = 7$ | *- p.n = 7 en q.n = 7* |

### 3.1.1 Lexicaal bereik / lexical scope

Als in een zekere scope een variabele wordt gereferenceerd (nog) niet is gedefinieerd, wordt in omliggende scopes “gezocht” naar een definitie van deze variabele.

```
F.1 local x;
F.2 local f;  f = function (i)
F.3     x = i + 5
F.4
F.5 f(5)                          - x = 10
```

..maar wanneer deze wel in de huidige scope bestaat, worden omliggende scopes “met rust gelaten”.

```
G.1 local x
G.2 local f
G.3 f = function (i)
G.4     local x
G.5     x = i + 5
G.6
G.7 f(5)                          - x heeft nog geen waarde
```

Telkens wanneer een functie wordt aangeroepen, wordt een *nieuwe scope* aangemaakt voor lokale variabelen. Variabelen van deze nieuwe scope kunnen later nog gereferenceerd worden, doordat bijvoorbeeld de functie een lokale functie teruggeeft.

```
H.1 local f
H.2 f = function (n) returns g
H.3     local g
H.4     g = function () returns n
H.5         n = n + 1
H.6
H.7 local c
H.8 c = f(5)                      - c() → 6, 7, 8, ...
```

maar dan wat beter geschreven, etc...

## 3.2 Grammatica

[...en vervolgens helemaal formeel – even uitleggen van BNF etc..]

# 4 Semantisch model

$$
\begin{array}{ll}
\mathbb{L} \stackrel{\text{def}}{=} \{(n, n) \in \mathbb{N}^2\} & \text{locaties van scopes en objecten} \\
\mathbb{F} \stackrel{\text{def}}{=} \mathit{Stm} \times \mathit{Id}_{\langle\rangle} \times (\mathit{Id} \cup \{\doteq\}) \times \mathbb{L} & \text{functies} \\
\mathbb{V} \stackrel{\text{def}}{=} \mathbb{L} \cup \mathbb{N} \cup \mathbb{F} & \text{waarden} \\
\mathbb{B} \stackrel{\text{def}}{=} \overset{\curvearrowleft}{\mathbb{V}}^{\mathit{Id}} & \text{binding-verzamelingen} \\
\mathbb{O} \stackrel{\text{def}}{=} \mathbb{B} \times (\mathbb{L} \cup \{\doteq\}) & \text{objecten} \\
\mathbb{S} \stackrel{\text{def}}{=} \mathbb{B} \times (\mathbb{L} \cup \{\doteq\}) & \text{scopes}
\end{array}
$$

[Stukje bij beetje het semantisch model opbouwen, terwijl we steeds redeneren waarom we dat zo doen..]

## 4.1 Scopes en lexical scope

[Hierarchieen, outer scopes, bindingen, “waarden” kort noemen maar uitstellen tot “Waarden: referenties en primitieven”]

## 4.2 Objecten en prototype overerving

[Graaf, bindingen, prototypen]

## 4.3 Waarden: referenties en primitieven

[Ze worden op dezelfde manier behandeld: objecten by-reference, dus de references zelf by-value, net als primitieven – vandaar dat ze in dezelfde verzameling waarden zitten.]

# 5 Natuurlijke Semantiek

Deze teksten zijn vooral bedoeld als "tekstvlees" (lorem ipsum's). We zullen axioma's en deductieregels introduceren waarmee we de relatie ($\longrightarrow$) definiëren, die de volgende signatuur heeft:

$$(\longrightarrow) \subseteq (Stm \times \mathbb{M} \times \mathbb{L} \times \mathbb{L}) \times \mathbb{M}$$

Wanneer we een uitspraak doen van de vorm:

$$\langle\, S \,\rangle_{m,\sigma,\tau} \longrightarrow m'$$

..dan bedoelen we daarmee dat:

$$\big(\,(S, m, \sigma, \tau), m'\,\big) \in (\longrightarrow)$$

Deze uitspraak moet je lezen als: "In de toestand met geheugen $m$, scope $\sigma$ en *this* object $\tau$, termineert het statement $S$, waarbij het resultaat-geheugen $m'$ is."

Een van deze axioma's [object], heeft betrekking tot de productieregel in de grammatica die de `object` "literal" introduceert.

$$\langle\, i \texttt{ object} \,\rangle_{m,\sigma,\tau} \longrightarrow m''$$

$$\begin{array}{ll} \textbf{desda} & \textsc{Find}_m(\sigma, i) = \sigma_{\text{def}} \\ & m' = m\big[\, \sigma_{\text{def}} \mapsto \big(b_{m(\sigma')}[i \mapsto \omega], p_{m(\sigma')}\big)\,\big] \\ & m'' = m'\big[\, \omega \mapsto \big(\varnothing, \perp\big)\,\big] \end{array}$$

Zoals vele axioma's en deductieregels heeft ook dit axioma een aantal voorwaarden waaraan moet worden voldaan. Deze staan eronder genoteerd, elk op een regel.

Wanneer bij een dergelijke opsomming van voorwaarden een nieuwe variabele wordt geïntroduceerd zoals hierboven, met de volgende vorm: **desda** $\square = \theta \ldots$; dan moet deze gelezen worden als: **desda** $\exists_\theta\big[\, \square = \theta \ldots \,\big]$.

$$\left\langle\, i \texttt{ clones } j \,\right\rangle_{m,\sigma,\tau} \longrightarrow m'$$

$$\begin{aligned} \textbf{desda} \quad & [\![\, i \,]\!]_{m,\sigma,\tau} = \omega_i \in \mathbb{L} \\ & [\![\, j \,]\!]_{m,\sigma,\tau} = \omega_j \in \mathbb{L} \\ & m' = m\big[\, \omega_i \mapsto \big(b_{m(\omega_i)}, \omega_j\big) \,\big] \end{aligned}$$

bla bla

$$\left\langle\, \texttt{local } i \,\right\rangle_{m,\sigma,\tau} \longrightarrow m'$$

$$\textbf{desda} \quad m' = m\big[\, \sigma \mapsto \big(b_{m(\sigma)}[i \mapsto \perp], p_{m(\sigma)}\big) \,\big]$$

bla bla

$$\left\langle\, \texttt{this}.s = e \,\right\rangle_{m,\sigma,\tau} \longrightarrow m'$$

$$\begin{aligned} \textbf{desda} \quad & \textsc{Trav}_m(\tau, s) = (\omega, i) \\ & [\![\, e \,]\!]_{m,\sigma,\tau} = v \\ & m' = m\big[\, \omega \mapsto \big(b_{m(\omega)}[i \mapsto v], p_{m(\omega)}\big) \,\big] \end{aligned}$$

bla bla

$$\left\langle\, i = e \,\right\rangle_{m,\sigma,\tau} \longrightarrow m'$$

$$\begin{aligned} \textbf{desda} \quad & \sigma_{\text{def}} = \textsc{Find}_m(\sigma, i) \\ & [\![\, e \,]\!]_{m,\sigma,\tau} = v \\ & m' = m\big[\, \sigma_{\text{def}} \mapsto \big(b_{m(\sigma_{\text{def}})}[i \mapsto v], p_{m(\sigma_{\text{def}})}\big) \,\big] \end{aligned}$$

bla bla

$$\left\langle\, i.s = e \,\right\rangle_{m,\sigma,\tau} \longrightarrow m'$$

$$\begin{aligned} \textbf{desda} \quad & \sigma_{\text{def}} = \textsc{Find}_m(\sigma, i) \\ & b_{m(\sigma_{\text{def}})}(i) = \omega \in \mathbb{L} \\ & \textsc{Trav}_m(\omega, s) = (\omega', j) \\ & [\![\, e \,]\!]_{m,\sigma,\tau} = v \\ & m' = m\big[\, \omega' \mapsto \big(b_{m(\omega')}[j \mapsto v], p_{m(\omega')}\big) \,\big] \end{aligned}$$

bla bla

$$\frac{\left\langle S_f \right\rangle_{m',\sigma_{f\text{new}},\omega'} \longrightarrow m''}{\left\langle i.s(e^*) \right\rangle_{m,\sigma,\tau} \longrightarrow m''}$$

**desda** $\sigma_{\text{def}} = \textsc{Find}_m(\sigma, i)$
$b_{m(\sigma_{\text{def}})}(i) = \omega \in \mathbb{L}$
$\textsc{Trav}_n(\omega, s) = (\omega', j)$
$(S_f, I_f, i_f, \sigma_{f\text{def}}) = f = b_{m(\omega')}(j)$
$\sigma_{f\text{new}} = \textsc{Next}_{\text{scope}}(m)$
$m' = m[\, \sigma_{f\text{new}} \mapsto (\llbracket\, e^* \,\rrbracket^*_{m,\sigma,\tau}(I_f), \sigma_{f\text{def}})\,]$

bla bla

$$\left\langle \texttt{skip} \right\rangle_{m,\sigma,\tau} \longrightarrow m$$

bla bla

$$\frac{\left\langle S_1 \right\rangle_{m,\sigma,\tau} \longrightarrow m' \qquad \left\langle S_2 \right\rangle_{m',\sigma,\tau} \longrightarrow m''}{\left\langle S_1; S_2 \right\rangle_{m,\sigma,\tau} \longrightarrow m''}$$

bla bla

$$\frac{\left\langle S_1 \right\rangle_{m,\sigma,\tau} \longrightarrow m'}{\left\langle \texttt{if}\,(b)\,\texttt{then}\,\{S_1\}\,\texttt{else}\,\{S_2\} \right\rangle_{m,\sigma,\tau} \longrightarrow m'}$$

**desda** $\llbracket\, b \,\rrbracket^{\mathrm{B}}_{m,\sigma,\tau} = \mathbf{T}$

bla bla

$$\frac{\left\langle S_2 \right\rangle_{m,\sigma,\tau} \longrightarrow m'}{\left\langle \texttt{if}\,(b)\,\texttt{then}\,\{S_1\}\,\texttt{else}\,\{S_2\} \right\rangle_{m,\sigma,\tau} \longrightarrow m'}$$

**desda** $\llbracket\, b \,\rrbracket^{\mathrm{B}}_{m,\sigma,\tau} = \mathbf{F}$

bla bla

$$\frac{\left\langle S_1 \right\rangle_{m,\sigma,\tau} \longrightarrow m' \qquad \left\langle \texttt{while}\,(b)\,\texttt{do}\,\{S_1\} \right\rangle_{m',\sigma,\tau} \longrightarrow m''}{\left\langle \texttt{while}\,(b)\,\texttt{do}\,\{S_1\} \right\rangle_{m,\sigma,\tau} \longrightarrow m''}$$

**desda** $\llbracket\, b \,\rrbracket^{\mathrm{B}}_{m,\sigma,\tau} = \mathbf{T}$

bla bla

$$\left\langle \texttt{while}\,(b)\,\texttt{do}\,\{S_1\} \right\rangle_{m,\sigma,\tau} \longrightarrow m$$

**desda** $\llbracket\, b \,\rrbracket^{\mathrm{B}}_{m,\sigma,\tau} = \mathbf{F}$

# 6 Case study: Wiskundige formulering

De vorm van bovenstaand semantisch model geeft een conceptueel sterk beeld van hoe de taal waarschijnlijk geimplementeerd zou worden in een compiler (modulo technische details, etc...). Het is ook vrijwel volgens bovenstaande semantiek dat aan informatica studenten object geörienteerde talen worden uitgelegd (referenties, primitieve waarden, ...). Je kunt echter ook semantisch model maken wat "wiskundiger van aard" is. Daar zal dit hoofdstuk over gaan.

Belangrijke informatie, zoals welk object van welk ander object een prototype is en hoe de scopes elkaar bevatten, maar ook welke informatie binnen een object zit opgeslagen en welke variabelen in een scope zijn gedefiniëerd, worden ditmaal door relaties bevat.

Rest nog identificatie van objecten en scopes, deze zal op eenzelfde manier worden behandeld als eerder de "locaties" binnen het geheugen: ze moeten identiek zijn, maar verder is het van geen belang hoe ze worden gerepresenteerd. We zullen daarom aannemen dat er twee verzamelingen identifiers zijn, $\mathbb{S}$ en $\mathbb{O}$, waarbij bovendien:

1. Voor beide verzamelingen $X \in \{\mathbb{S}, \mathbb{O}\}$, bestaat er een functie $\textsc{Next} : \mathcal{P}(X) \to X$, zdd $\textsc{Next}(Y) \notin Y$ voor alle $Y \subseteq X$.
2. $\mathbb{S} \cap \mathbb{O} = \varnothing$ (Deze eis is niet een technische benodigdheid, maar geeft wel aan dat de semantiek van deze twee soorten identifiers nu eenmaal anders is.)

De prototype hiërarchie, eveneens de scope hiërarchie, worden natuurlijk heel goed weergegeven door partiële ordeningen. Deze twee relaties zullen zullen we als $\sqsubset^p$ en $\sqsubset^s$, respectievelijk, noteren, en aannemen:

1. $\sqsubset^p$ is een (tweestemmige) partiële ordening op $\mathbb{O}$
2. $\sqsubset^s$ is een (tweestemmige) partiële ordening op $\mathbb{S}$

Vervolgens introduceren we de relatie $attr \subseteq \mathbb{O} \times Id \times (\mathbb{O} \cup \mathbb{P})$. De uitspraak "$p[i] = v$" moet worden gelezen als: $(p, i, v) \in attr$. Deze relatie wordt eveneens gebruikt om de "inhoud" van objecten weer te geven, als de graafstructuur tussen objecten.

Soortgelijk definiëren we de relatie $def \subseteq \mathbb{S} \times Id \times (\mathbb{O} \cup \mathbb{P})$, waarbij de uitspraak "$\sigma[i] = v$" moet worden gelezen als: $(\sigma, i, v) \in def$.

Ten alle tijde moeten de prototype en scope hiërarchieën, de inhoud van objecten en scopes, en de laatste indentificaties van objecten en scopes worden bijgehouden, en dit vormt dan het "geheugen", of de "toestand": $s = (attr, def, \sqsubset^p, \sqsubset^s, p_{next}, \sigma_{next})$.

$$
\begin{aligned}
\langle \texttt{skip}, s, \tau, \sigma \rangle &\longrightarrow s \\
\langle x = e, s, \tau, \sigma \rangle &\longrightarrow s[\ \sigma_d[x] = e\ ] \\
&\qquad \textbf{desda}\ \sigma_d = \max\{\, \sigma' \in \mathbb{S} \mid \sigma'[x] \wedge \sigma \sqsubseteq^s \sigma' \,\} \\
\langle x\ \texttt{object}, \tau, \sigma \rangle &\longrightarrow s[\ \sigma_d[x] = p\ ][\ p_{next} \mapsto p + 1\ ] \\
&\qquad \textbf{desda}\ \sigma_d = \max\{\, \sigma' \in \mathbb{S} \mid \sigma'[x] \wedge \sigma \sqsubseteq^s \sigma' \,\} \\
&\qquad \textbf{en}\ p = p_{next_s}
\end{aligned}
$$