Reflection Languages and Lambda Calculus

Quick Reference

Henk Barendregt
Nijmegen Universiteit

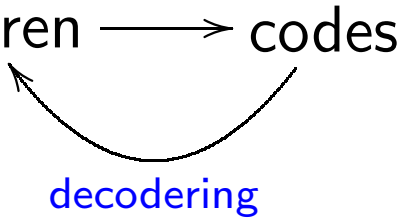
Schakelblok Informatica
Winter 2004

Reflectie (1)

Actoren: domein van actieve objecten

Codes: zelf meestal passief; decodering tot actoren
niet noodzakelijkerwijs in domein

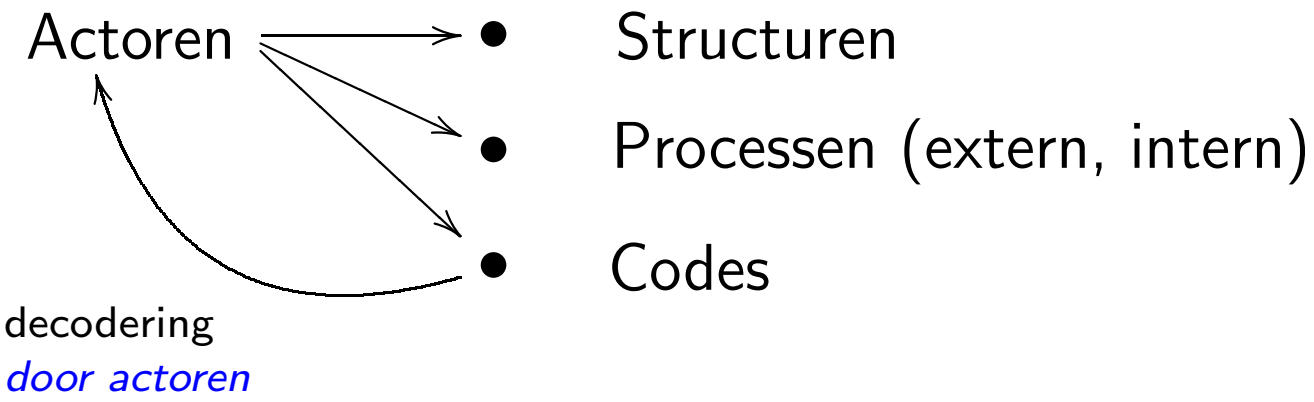
Interactie:



Onderwerp	Actoren	Codes
Biologie	eiwitten	DNA
Taal	zinsdelen	‘citaten’
Wiskunde	uitspraken	‘citaten’
Informatica	berekenbare functies	programma’s
Meditatie	bewustzijnsinhouden	opmerkzaamheid

Reflectie (2)

Globale terugkoppeling



Kracht en keerzijde van reflectie

Onderwerp	'nuttig' effect	'ongewenst' effect
Biologie	leven	virussen
Taal	homo sapiens	paradoxen
Wiskunde	wetenschap	essentiële onbewijsbaarheid
Informatica	IT	essentiële onberekenbaarheid
Meditatie	zuivering	verwarring

Talen (1)

Een *alfabet* Σ is een verzameling van symbolen. Een *woord* (string) over Σ is een eindig rijtje elementen uit Σ. De verzameling Σ^* bestaat uit alle woorden over Σ. Een taal over Σ is een deelverzameling $L \subseteq \Sigma^*$.

Operaties op woorden. Laat $a \in \Sigma$ een symbool, $u, v \in \Sigma^*$ woorden en $L, L_1, L_2 \subseteq \Sigma^*$ talen zijn. Dan is $ua \in \Sigma^*$ het woord u met erachter het symbool a; $uv \in \Sigma^*$ is het woord dat ontstaat door achter u het woord v te plaatsen; $L_1 \cup L_2 \subseteq \Sigma^*$ is de taal bestaande uit de woorden in L_1 tezamen met die uit L_2. L^* bestaat uit de collectie verkregen door een willekeurig aantal woorden uit L achter elkaar te plakken.

Reguliere expressies over Σ:

$$\boxed{\texttt{exp} ::= \emptyset \mid \epsilon \mid \texttt{a} \mid \texttt{exp exp} \mid \texttt{exp} \cup \texttt{exp} \mid (\texttt{exp})^*}$$

Voor een reguliere expressie e definieren we de taal $L(e)$ over Σ:

$$\begin{aligned} L(\emptyset) &= \emptyset \\ L(\epsilon) &= \{\epsilon\} \\ L(a) &= \{a\} \\ L(e_1 e_2) &= L(e_1) L(e_2) \\ L(e_1 \cup e_2) &= L(e_1) \cup L(e_2) \\ L((e)^*) &= L(e)^* \end{aligned}$$

Een taal L heet *regulier* als $L = L(e)$ voor een reguliere expressie e.

Talen (2)

Een *context vrije taal* L over Σ wordt gegenereerd door productieregels van de vorm

$$X \to w$$

met $X \in V$ en $w \in (\Sigma \cup V)^*$. De elementen van V heten hulpsymbolen. Er is een $S \in V$ (start). L wordt uit deze regels gekregen m.b.v. de relatie $\Rightarrow^*$ gedefinieerd alsvolgt (er geldt $u, v, w, x, y \in (\Sigma \cup V)^*$):

$$\begin{array}{rcl} X \to w & \Rightarrow & xXy \Rightarrow^* xwy \\ u \Rightarrow^* v,\ v \Rightarrow^* w & \Rightarrow & u \Rightarrow^* w \end{array}$$

Tenslotte is

$$L = \{w \in \Sigma^* \mid S \Rightarrow^* w\}.$$

Notatie. $X \to w_1 \mid w_2$ staat voor

$$\begin{array}{rcl} X & \to & w_1 \\ X & \to & w_2 \end{array}$$

De *context gevoelige talen* gaan uit van productie regels van de vorm

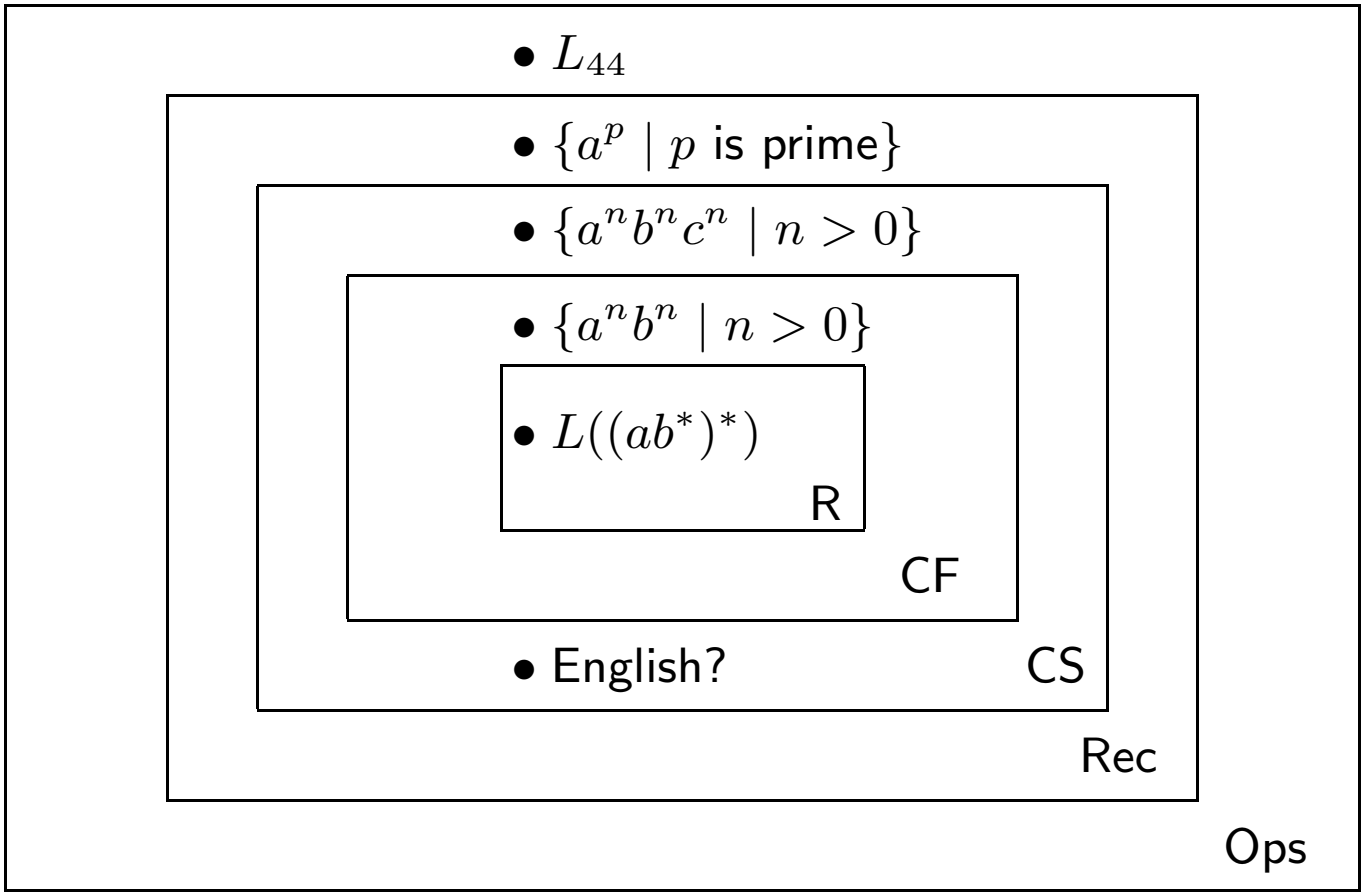
$$xXy \to xwy,$$

waarbij $w \neq \epsilon$.

Bij de *opsombare talen* daarentegen mag $w = \epsilon$.

Talen (3): Chomsky hiërarchie

Een taal L heet recursief (berekenbaar) als zowel L als $\overline{L} = \Sigma^* - L$ opsombaar zijn. Laten R, CF, CS, Rec, Ops notaties zijn voor de reguliere, context vrije, context gevoelige, recursieve en opsombare talen, respectivelijk. Dan R $\subseteq$ CF $\subseteq$ CS $\subseteq$ Rec $\subseteq$ Ops.



The Chomsky hierarchy

Talen (4): Klein deel van het Engels als CF taal

$$
\begin{aligned}
S = \langle sentence \rangle &\rightarrow \langle noun-phrase \rangle \langle verb-phrase \rangle . \\
\langle sentence \rangle &\rightarrow \langle noun-phrase \rangle \langle verb-phrase \rangle \langle object-phrase \rangle . \\
\langle noun-phrase \rangle &\rightarrow \langle name \rangle \mid \langle article \rangle \langle noun \rangle \\
\langle name \rangle &\rightarrow John \mid Jill \\
\langle noun \rangle &\rightarrow bicycle \mid mango \\
\langle article \rangle &\rightarrow a \mid the \\
\langle verb-phrase \rangle &\rightarrow \langle verb \rangle \mid \langle adverb \rangle \langle verb \rangle \\
\langle verb \rangle &\rightarrow eats \mid rides \\
\langle adverb \rangle &\rightarrow slowly \mid frequently \\
\langle adjective-list \rangle &\rightarrow \langle adjective \rangle \langle adjective-list \rangle \mid \epsilon \\
\langle adjective \rangle &\rightarrow big \mid juicy \mid yellow \\
\langle object-phrase \rangle &\rightarrow \langle adjective-list \rangle \langle name \rangle \\
\langle object-phrase \rangle &\rightarrow \langle article \rangle \langle adjective-list \rangle \langle noun \rangle
\end{aligned}
$$

Combinatory Logic (1)

$\Sigma_{\mathbf{CL}} = \{\mathbf{I}, \mathbf{K}, \mathbf{S}, x, ',), (\}$

$\langle$`constant`$\rangle$	$\rightarrow$	$\mathbf{I} \mid \mathbf{K} \mid \mathbf{S}$
$\langle$`variable`$\rangle$	$\rightarrow$	$x \mid \langle$`variable`$\rangle'$
$\langle$`term`$\rangle$	$\rightarrow$	$\langle$`constant`$\rangle \mid \langle$`variable`$\rangle \mid (\langle$`term`$\rangle\ \langle$`term`$\rangle)$

We use	$x, y, z, \ldots, x_0, y_0, z_0, \ldots, x_1, y_1, z_1, \ldots$	to denote variables
	$P, Q, R, \ldots, X, Y, Z, \ldots$	to denote terms
	c	to denote a constant $\mathbf{I}, \mathbf{K}$ or $\mathbf{S}$
	$PQ_1 \ldots Q_n$	to denote $(..((PQ_1)Q_2) \ldots Q_n)$

The theory $\mathbf{CL}$ consists of statements $P =_{\mathbf{CL}} Q$ (or just $P = Q$) axiomatized by

$\mathbf{I}P$	$=_{\mathbf{CL}}$	P
$\mathbf{K}PQ$	$=_{\mathbf{CL}}$	P
$\mathbf{S}PQR$	$=_{\mathbf{CL}}$	$PR(QR)$

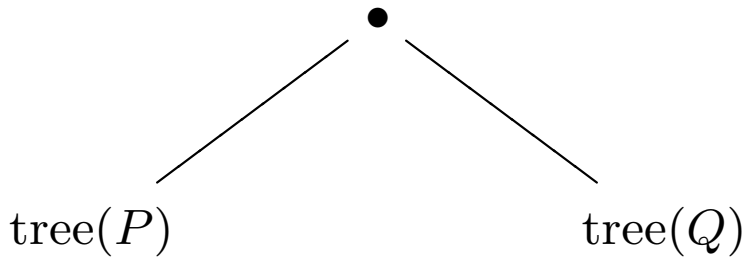
Define $\mathbf{D} \equiv \mathbf{SII}$, $\mathbf{B} \equiv \mathbf{S(KS)K}$, $\mathbf{L} \equiv \mathbf{D(BDD)}$

Then

$\mathbf{D}x$	$=_{\mathbf{CL}}$	xx	(doubling)
$\mathbf{B}fgx$	$=_{\mathbf{CL}}$	$f(gx)$	(composition)
$\mathbf{L}$	$=_{\mathbf{CL}}$	$\mathbf{LL}$	(self-doubling, life!)

Combinatory Logic (2)

The tree of a $\mathbf{CL}$-term

M	$\mathrm{tree}(M)$
x	x
c	c
(PQ)	

Making abstractions in $\mathbf{CL}$.

P	$\lambda^* x.P$
x	$\mathbf{I}$
y	$\mathbf{K}y$
c	$\mathbf{K}c$
QR	$\mathbf{S}(\lambda^* x.Q)(\lambda^* x.R)$

or better

P	$\lambda^* x.P$
x	$\mathbf{I}$
$P,\ x \notin \mathrm{FV}(P)$	$\mathbf{K}P$
$QR,\ x \in \mathrm{FV}(P)$	$\mathbf{S}(\lambda^* x.Q)(\lambda^* x.R)$

Lambda calculus (1)

$\Sigma_\lambda = \{x, ', \lambda,), (\}$ (just one symbol more than the DNA alphabet)

Lambda terms

$$\begin{array}{lcl} \langle \texttt{variable} \rangle & \rightarrow & x \mid \langle \texttt{variable} \rangle' \\ \langle \texttt{term} \rangle & \rightarrow & \langle \texttt{variable} \rangle \mid (\lambda \langle \texttt{variable} \rangle \ \langle \texttt{term} \rangle) \mid (\langle \texttt{term} \rangle \ \langle \texttt{term} \rangle) \end{array}$$

Axiom ($M[\vec{x} := \vec{N}]$ denotes the result of substituting the $\vec{N}$ for the $\vec{x}$ in M)

$$(\lambda x.M)N =_\lambda M[x := N] \qquad (\beta\text{-rule})$$

Conventions. $x, y, z, \ldots$ and $M, N, L, \ldots$ denote arbitrary variables and terms respectively. $M \equiv N$ means that M, N are literally the same lambda term.

$$\begin{array}{lcll} MN_1 \ldots N_n & \equiv & (..((MN_1)N_2) \ldots N_n) & \text{association to the left} \\ \lambda x_1 \ldots x_n.M & \equiv & (\lambda x_1(\lambda x_2(..(\lambda x_n(M))..)))) & \text{association to the right} \\ \lambda x_1 \ldots x_n.M & \equiv & \lambda y_1 \ldots y_n.M[x_1, \ldots, x_n := y_1, \ldots, y_n] & \text{renaming 'bound' variables} \end{array}$$

One has

$$\begin{array}{lcl} (\lambda \vec{x}.M)\vec{x} & = & M \\ (\lambda \vec{x}.M)\vec{N} & = & M[\vec{x} := N] \end{array}$$

We like to abbreviate an expression like $bX(c(Xa))$ as $C[a, b, c, X]$. Such an expression $C[\ldots]$ is called a *context*. $C[y_1, \ldots, y_n, x]$ will be abbreviated as $C[\vec{y}, x]$.

Lambda calculus (2)

THEOREM. For every lambda term F there exists a lambda term X such that

$$FX =_\lambda X.$$

Given F. Define $W \equiv \lambda x.F(xx)$ and $X \equiv WW$. Then

$$\begin{array}{lll} X & \equiv & WW \\ & \equiv & (\lambda x.F(xx))W \\ & =_\lambda & F(WW) \\ & \equiv & FX. \blacksquare \end{array}$$

COROLLARY. Given a context $C[\vec{y}, x]$. Then there exists a term X such that

$$X\vec{y} =_\lambda C[\vec{y}, X].$$

Hence for all terms $\vec{P} =_\lambda P_1, \ldots, P_n$

$$X\vec{P} =_\lambda C[\vec{P}, X].$$

APPLICATIONS. There are L,O,P,Z such that L$=_\lambda$ LL, O$x =_\lambda$ O, P$=_\lambda$ PI, Z$x =_\lambda x$Z.

Let $F^0A \equiv A$, $F^{n+1}A \equiv F(F^nA)$; define the Church numerals $\mathsf{c}_n \equiv \lambda fa.f^na$. Then

$$(\lambda nmfa.nf(mfa))\,\mathsf{c}_n\,\mathsf{c}_m =_\lambda \mathsf{c}_{n+m}.$$

Arbitrary computable operations on numbers can be lambda defined similarly.

Reflection in lambda calculus (1)

DEFINITION. A redex is a lambda term of the form $(\lambda x.M)N$.

A term M is a normal form (nf) if M does not contain a redex.

A term M has a nf if $M =_\lambda N$ and N is a nf.

FACT. Every lambda term has at most one nf.

Hence $x \neq_\lambda x'$, K $\neq_\lambda$ I.

COROLLARY. There are no terms P_1, P_2 such that

$P_1(xy) = x$ or $P_2(xy) = y$.

DATA TYPES.

`nat`	$\rightarrow$	`z | s(nat)`
`tree`	$\rightarrow$	`b | P tree tree`
`ltree`	$\rightarrow$	`L var | P ltree ltree | !ltree`
`var`	$\rightarrow$	`x | var'`

Böhm-Berarducci (BB) representation of first two data types.

$\lambda sz.s^n z$ (Church numerals); $\lambda bP.Pb(Pbb)$, $\lambda bP.P(Pbb)(Pbb)$.

We have seen the representation of addition on `nat`.

Mirroring on `tree`: $F_{\text{mirror}} \equiv \lambda tbP.tbP'$, where $P' \equiv \lambda ab.Pba$.

Then $F_{\text{mirror}}(\lambda bP.Pb(Pbb)) =_\lambda \lambda bP.P(Pbb)b$.

Reflection in lambda calculus (2)

Tuples and projections: $\langle M_1, \ldots, M_n\rangle \equiv \lambda z.zM_1 \ldots M_n.$
$U_i^n \equiv \lambda x_1 \ldots x_n.x_i.$

Then $\langle M_1, \ldots, M_n\rangle U_i^n =_\lambda M_i.$

Böhm-Piperno-Guerrini (BPG) representation of third data type.

Define $F_L \equiv \lambda xe.eU_1^3xe;$ or more mnemonical $F_Lx =_\lambda \lambda e.eU_1^3xe;$
$F_P \equiv \lambda xye.eU_2^3xye;$ $F_Pxy =_\lambda \lambda e.eU_2^3xye;$
$F_! \equiv \lambda xe.eU_3^3xe.$ $F_!x =_\lambda \lambda e.eU_3^3xe.$

Now define $\ulcorner Lx\urcorner =_\lambda F_Lx;$ or in nf $\ulcorner Lx\urcorner \equiv \lambda e.eU_1^3xe;$
$\ulcorner Pt_1t_2\urcorner =_\lambda F_P\ulcorner t_1\urcorner\ulcorner t_2\urcorner;$ $\ulcorner Pt_1t_2\urcorner \equiv \lambda e.eU_2^3\ulcorner t_1\urcorner\ulcorner t_2\urcorner e;$
$\ulcorner !t\urcorner =_\lambda F_!\ulcorner t\urcorner.$ $\ulcorner !t\urcorner \equiv \lambda e.eU_3^3\ulcorner t\urcorner e.$

PROPOSITION. Let A_1, A_2, A_3 be given lambda terms. Then there exists a H such that

$H(F_Lx) =_\lambda A_1xH;$ Hint. Try $H \equiv \langle\langle B_1, B_2, B_3\rangle\rangle.$
$H(F_Pxy) =_\lambda A_2xyH;$
$H(F_!x) =_\lambda A_3xH.$

APPLICATION. There exists an H that erases the !'s in an `ltree`.

$H(F_Lx) =_\lambda F_Lx,$ take $A_1 \equiv \lambda xh.F_Lx;$
$H(F_Pxy) =_\lambda F_P(Hx)(Hy),$ take $A_2 \equiv \lambda xyh.F_P(hx)(hy);$
$H(F_!x) =_\lambda Hx,$ take $A_3 \equiv \lambda xh.hx.$

Reflection in lambda calculus (3)

Coding lambda terms as other lambda terms in nf (Mogensen).

$$\begin{array}{rcll} \ulcorner x \urcorner & \equiv & \lambda e.eU_1^3 xe & =_\lambda \quad F_L x; \\ \ulcorner MN \urcorner & \equiv & \lambda e.eU_2^3 \ulcorner M \urcorner \ulcorner N \urcorner e & =_\lambda \quad F_P \ulcorner M \urcorner \ulcorner N \urcorner; \\ \ulcorner \lambda x.M \urcorner & \equiv & \lambda e.eU_3^3 (\lambda x.\ulcorner M \urcorner) e & =_\lambda \quad F_! (\lambda x.\ulcorner M \urcorner). \end{array}$$

By the above proposition there exists a lambda term E (self-interpreter) such that

$$\begin{array}{rcl} \mathsf{E}\ulcorner x \urcorner & =_\lambda & x; \\ \mathsf{E}\ulcorner MN \urcorner & =_\lambda & \mathsf{E}\ulcorner M \urcorner (\mathsf{E}\ulcorner N \urcorner); \\ \mathsf{E}\ulcorner \lambda x.M \urcorner & =_\lambda & \lambda x.(\mathsf{E}\ulcorner M \urcorner). \end{array}$$

Hence for all lambda terms M one has

$$\mathsf{E}\ulcorner M \urcorner =_\lambda M.$$

Following the construction one can take $\mathsf{E} \equiv \langle\langle \mathsf{K}, \mathsf{S}, \mathsf{C} \rangle\rangle$.

There exists lambda terms P_1, P_2 such that

$$P_1 \ulcorner MN \urcorner =_\lambda \ulcorner M \urcorner \text{ and } P_2 \ulcorner MN \urcorner =_\lambda \ulcorner N \urcorner.$$

There exists a lambda term Q such that

$$Q \ulcorner MN \urcorner \ulcorner L \urcorner =_\lambda \ulcorner ML \urcorner.$$

Reflection revisited

The last slide shows that reflection gives power. We can select from the code of a term (but not from the term itself) or we can replace part of it by the code of another term.

THEOREM. For all lambda terms F there is a lambda term X such that

$$F\ulcorner X\urcorner =_\lambda X.$$

APPLICATION. There is a term H such that

$$\begin{array}{lcll} H\,\mathsf{c}_n & = & \mathsf{c}_{3n} & \text{if } n \text{ is even;} \\ H\,\mathsf{c}_n & = & G\ulcorner H\urcorner\,\mathsf{c}_n & \text{else.} \end{array}$$

Typical use of reflection actually happens during translation (so called compiling) of higher programming languages into machine code. Often the compiler of the higher programming language is written in that language itself. In order to run that compiler the first time, one needs an older (usually less efficient) compiler in another language.