From λ-terms to combinators

The translation $(\_)_{cl}$: set of λ-terms $\longrightarrow$ set of CL terms is defined inductively by

$$\begin{aligned}(x)_{cl} &\equiv x\\ (MN)_{cl} &\equiv (M)_{cl}(N)_{cl}\\ (\lambda x.M)_{cl} &\equiv \lambda^* x.(M)_{cl}\end{aligned}$$

Additional combinators

The combinators **B** and **C** with reduction rules

$$\mathbf{B}PQR \to P(QR) \qquad \mathbf{C}PQR \to PRQ.$$

Remark

In fact, **B** and **C** can be expressed using the original Cl terms

$$\begin{aligned}\mathbf{B} &\equiv \mathbf{S}(\mathbf{KS})\mathbf{K}\\ \mathbf{C} &\equiv \mathbf{S}(\mathbf{BBS})(\mathbf{KK}).\end{aligned}$$

Summary of implementation methods

1. ISWIM $\longrightarrow$ λ-calculus $\longrightarrow$ SECD machine;
2. λ-calculus $\longrightarrow$ de Bruijn notation;
3. λ-calculus $\longrightarrow$ combinators $\longrightarrow$ graph reduction.

Abstraction on combinators

Definition

The operation $\lambda^* x$ on combinators, where x is a variable, is defined recursively as follows:

$\lambda^* x.x$ is **I**;

$\lambda^* x.P$ is $\mathbf{K}P$ if $x \notin \mathrm{FV}(P)$;

$\lambda^* x.PQ$ is $\mathbf{S}(\lambda^* x.P)(\lambda^* x.Q)$.

Notation $\lambda^* x_1 \ldots x_n.P \equiv \lambda^* x_1.(\lambda^* x_2. \ldots .\lambda^* x_n.P \ldots))$.

Alternative abstraction on combinators

The operation $\lambda^{\mathsf{T}}x$ on combinators, where x is a variable, is defined recursively as follows:

$\lambda^{\mathsf{T}}x.x$ is **I**;

$\lambda^{\mathsf{T}}x.P$ is **K**P if $x \notin \mathsf{FV}(P)$;

$\lambda^{\mathsf{T}}x.Px$ is P if $x \notin \mathsf{FV}(P)$;

$\lambda^{\mathsf{T}}x.PQ$ is **B**$P(\lambda^{\mathsf{T}}x.Q)$ if $x \notin \mathsf{FV}(P)$ and $Q \not\equiv x$;

$\lambda^{\mathsf{T}}x.PQ$ is $C(\lambda^{\mathsf{T}}x.P)Q$ if $x \notin \mathsf{FV}(Q)$;

$\lambda^{\mathsf{T}}x.PQ$ is **S**$(\lambda^{\mathsf{T}}x.P)(\lambda^{\mathsf{T}}x.Q)$ is $x \in \mathsf{FV}(P) \cap \mathsf{FV}(Q)$.