Formal and Constructive Theory of Computation

Initial Bachelor Seminar Talk

Fabian Kunze
Advisor: Prof. Dr. Gert Smolka

SAARLAND UNIVERSITY
COMPUTER SCIENCE

24.04.2015

Content

Previous Work

J. Roger Hindley and Jonathan P. Seldin.
Introduction to Combinators and λ–Calculus,
Cambridge University Press, 1986.

Yannick Forster
A Formal and Constructive Theory of Computation
Bachelor thesis, Saarland University, 2014

L: Call-by-Value λ-Calculus

$$s, t \ ::= \ x \ \mid \ \lambda x.s \ \mid \ s \ t \quad (x \in \mathbb{N})$$

$$\frac{}{(\lambda x.s)\ (\lambda y.t) >_L s^x_{\lambda y.t}}$$

$$\frac{s >_L s'}{s\ t >_L s'\ t} \qquad \frac{t >_L t'}{s\ t >_L s\ t'}$$

$$\begin{aligned} x^x_u &= u \\ x^y_u &= x && \text{if } x \neq y \\ (\lambda x.s)^x_u &= \lambda x.s \\ (\lambda x.s)^y_u &= \lambda x.(s^y_u) && \text{if } x \neq y \\ (s\ t)^x_u &= s^x_u\ t^x_u \end{aligned}$$

- evaluates abstraction only if argument is irreducible
- strength (and complexity) is in substitution
- Goal: more basic system that can "compute the same"

SK Combinator Calculus

$$s, t \ ::= \ x \ \mid \ \mathsf{S} \ \mid \ \mathsf{K} \ \mid \ s \ t \quad (x \in \mathbb{N})$$

$$\frac{}{\mathsf{K}\ s\ t >_{SK} s} \qquad \frac{}{\mathsf{S}\ s\ t\ u >_{SK} (s\ u)\ (t\ u)} \qquad \frac{s >_{SK} s'}{s\ t >_{SK} s'\ t} \qquad \frac{t >_{SK} t'}{s\ t >_{SK} s\ t'}$$

- also called *SKI*, but combinator I can be defined:

$$\mathsf{I} := \mathsf{S}\ \mathsf{K}\ \mathsf{K}$$

$$\mathsf{I}\ x = \mathsf{S}\ \mathsf{K}\ \mathsf{K}\ x >_{SK} \mathsf{K}\ x\ (\mathsf{K}\ x) >_{SK} x$$

- really short confluence proof in Coq

SK Combinator Calculus

SK can "simulate" substitution

Example

$\lambda x.(x\ y) \sim \mathsf{S\ I\ (K}\ y)$:

$$(\lambda x.(x\ y))\ z >_L z\ y$$

$$\mathsf{S\ I\ (K}\ y)\ z > (\mathsf{I}\ z)\ (\mathsf{K}\ y\ z) >^* z\ y$$

- S : 'push' argument down in application
- K : discard 'pushed' argument
- I : take 'pushed' argument

- But *SK* can not restrict reduction until arguments are irreducible

SK_v: Call-by-Value SK

$$s, t \ ::= \ x \ \mid \ \mathsf{K} \ \mid \ \mathsf{S} \ \mid \ s \ t \quad (x \in \mathbb{N})$$

$$\mathit{Val}\text{: } s, t \ ::= \ x \ \mid \ \mathsf{K} \ \mid \ \mathsf{K} \ s \ \mid \ \mathsf{S} \ \mid \ \mathsf{S} \ s \ \mid \ \mathsf{S} \ s \ t \quad , x \in \mathbb{N}$$

$$\frac{s, t \in \mathit{Val}}{\mathsf{K} \ s \ t > s} \qquad \frac{s, t, u \in \mathit{Val}}{\mathsf{S} \ s \ t \ u > s \ u \ (t \ u)} \qquad \frac{s > s'}{s \ t > s' \ t} \qquad \frac{t > t'}{s \ t > s \ t'}$$

- values are irreducible
- closed, irreducible terms are values
- FV (s): free variables in s
- uniform confluent (like L, but not SKI and λ-calculus):

$$s > t_1 \wedge s > t_2 \Longrightarrow t_1 = t_2 \vee (\exists u : t_1 > u \wedge t_2 > u)$$

(Pseudo)-Abstractions in SK_v

$$
\begin{aligned}
[x].s &:= \mathsf{K}\ s && \text{if } x \notin \mathrm{FV}(s) \\
[x].x &:= \mathsf{I} \\
[x].(s\ t) &:= \mathsf{S}\ ([x].s)\ ([x].t) && \text{if } x \in \mathrm{FV}(s\ t)
\end{aligned}
$$

- behaves like abstraction in L:
$$s, t \in \mathit{Val} \Longrightarrow ([x].s)\ t >^* s_t^x$$
- compatible with substitution:
$$x \neq y \wedge y \notin \mathrm{FV}(t) \Longrightarrow ([y].s)_t^x = [y].(s_t^x)$$
- $s \in \mathit{Val} \Longrightarrow [x].s \in \mathit{Val}$
- but we can reduce 'under' the abstraction:
$[x].(\mathsf{I}\ \mathsf{I}) = \mathsf{I}\ \mathsf{I} >^* \mathsf{I} = [x].\mathsf{I}$

idea: use S to separate applications

Isomorphic Translation $L \to SK_v$

$$\begin{array}{cc} \mathrm{H}x := x & \mathrm{H'}x := \mathrm{K}\ x \\ \mathrm{H}(s\ t) := (\mathrm{H}s)\ (\mathrm{H}t) & \mathrm{H'}(s\ t) := \mathrm{S}\ (\mathrm{H'}s)\ (\mathrm{H'}t) \\ \mathrm{H}(\lambda x.s) := \mathrm{S}\ ([x].\mathrm{H'}s)\ \mathrm{I} & \mathrm{H'}(\lambda x.s) := \mathrm{K}\ (\mathrm{S}\ ([x].\mathrm{H'}s)\ \mathrm{I}) \end{array}$$

- H' is an S-lifted version of H
- $\mathrm{H'}s \in \mathit{Val}$
- $u \in \mathit{Val} \Longrightarrow (\mathrm{H'}s)\ u >^* \mathrm{H}s$
- $s >_L t \Longrightarrow \mathrm{H}s >^+ \mathrm{H}t$
- $s \in L$ is irreducible $\Longleftrightarrow \mathrm{H}s \in \mathit{Val}$

SK_v-Decidability vs. L-Decidability

P is L-decidable iff there exists an closed L-term u s.t. for all s:

$$P(s) \wedge u \ulcorner s \urcorner >_L^* \text{true}$$
$$\vee \neg P(s) \wedge u \ulcorner s \urcorner >_L^* \text{false}$$

P is SK_v-decidable iff there exists an closed SK_v-term u s.t. for all s:

$$P(s) \wedge u\ (\mathrm{H} \ulcorner s \urcorner) >^* \mathrm{H}\ \text{true}$$
$$\vee \neg P(s) \wedge u\ (\mathrm{H} \ulcorner s \urcorner) >^* \mathrm{H}\ \text{false}$$

- L-decidability $\Longrightarrow$ SK_v-decidability:
  For an L-decider u, Hu is the corresponding SK_v-decider
- SK_v-decidability $\Longrightarrow$ L-decidability:
  Encode SK_v in L as Scott-encoded datatype and internalize an step-function. Uniform confluence simplifies proof.

Further Results and Outlook

further results:

- L vs. L_n (De Bruijn indices vs. named variables)
- Tactic for faster normalization in L (using reflection)
- nice, short confluence proof for SK

outlook:

- 'L-decidability $\Longrightarrow$ SK_v-decidability' in coq
- may improve normalization: environments instead of substitution
- weaker notion of L-decidability not implying Coq-decidability
- formalize and prove that L and SK_v can compute the same functions
- use SK_v to show that other systems are L-complete
- show that H maps exactly α-equivalent to the same SK_v-term

5 Appendix

- SK-confluence using Parallel Reduction

SK-confluence using Parallel Reduction

$$\frac{}{s \gg s} \qquad \frac{}{\mathsf{K}\ s\ t \gg s} \qquad \frac{}{\mathsf{S}\ s\ t\ u \gg s\ u\ (t\ u)} \qquad \frac{s \gg s' \qquad t \gg t'}{s\ t \gg s'\ t'}$$

- $> \quad \subsetneq \quad \gg \quad \subsetneq \quad >^*$
- $s >^* t \iff s \gg^* t$
- diamond property:

$$s \gg t \wedge s \gg t' \Longrightarrow \exists u, t \gg u \wedge t' \gg u$$

- confluence for $\gg \Longrightarrow$ confluence for $>$