Computational Logic and Applications, CLA '05 DMTCS proc. **AF**, 2006, 87–96

Classical Combinatory Logic

Karim Nour[1†]

[1] *LAMA - Equipe de logique , Université de Savoie , F-73376 Le Bourget du Lac, France*

Combinatory logic shows that bound variables can be eliminated without loss of expressiveness. It has applications both in the foundations of mathematics and in the implementation of functional programming languages. The original combinatory calculus corresponds to minimal implicative logic written in a system "à la Hilbert". We present in this paper a combinatory logic which corresponds to propositional classical logic. This system is equivalent to the system λ^{Sym}_{Prop} of Barbanera and Berardi.

Keywords: Combinatory logic, Lambda-calculus, Propositional classical logic

1 Introduction

Combinatory logic started with a paper by Schönfinkel (1924). The aim was an elimination of bound variables. He proved that it is possible to reduce the logic to a language consisting of one constructor (the application) and some primitive constants. This work was continued by Curry and Feys (1958) who introduced the syntax of the terms of combinatory logic. At about the same time, Church (1941) introduced the lambda-calculus as a new way to study the concept of rule. Originally his purpose was to provide a foundation for mathematics. Combinatory logic and lambda-calculus, in their type-free version, generate essentially the same algebraic and logic structures. The original combinatory calculus corresponds to minimal implicative logic presented in a system "à la Hilbert". The codings between combinatory logic and simply typed calculus preserve types. Research on combinatory logic has been continued essentially by Curry's students, Hindley and Seldin (1986).

Since it has been understood that the Curry-Howard isomorphism relating proofs and programs can be extended to classical logic, various systems have been introduced: the λ_c-calculus (Krivine (1994)), the λ_{exn}-calculus (DeGroote (1995)), the $\lambda\mu$-calculus (Parigot (1992)), the λ^{Sym}-calculus (Barbanera and Berardi (1994)), the λ_Δ-calculus (Rehof and Sorensen (1994)), the $\overline{\lambda}\mu\tilde{\mu}$-calculus (Curien and Herbelin (2000)), the dual calculus (Wadler (2005)) ... All these calculi are based on logical systems presented either in natural deduction or in sequent calculus.

We wish to define a combinatory calculus which corresponds to classical logic presented "à la Hilbert". There are two ways to define such a calculus:

- Add new combinators for the axioms which define classical logic over minimal logic and give the corresponding reduction rules.
- Code by combinators an existing calculus based on classical logic.

[†]Karim.Nour@univ-savoie.fr

The first way gives a very "artificial" solution. The reduction rules for the new combinators are rather complicated. For the second way, it is necessary to choose a system such that the reduction rules erase the abstractors (i.e. the right-hant side of the reduction rules should not introduce new abstractions). One of these calculi is the λ^{Sym}-calculus of Barbanera and Berardi.

We present in this paper the λ^{Sym}-calculus and the new combinatory calculus CCL. We also explain how to encode each calculus into the other.

The paper is organized as follows. In section 2, we give the syntax of the terms and the reduction rules of the system λ^{Sym}_{Prop}. We introduce, in section 3, the syntax of the terms and the reduction rules of the system CCL. We encode, in section 4, the system λ^{Sym}_{Prop} into the system CCL and we encode, in section 5, the system CCL into the system λ^{Sym}_{Prop}. We conclude with some future work.

2 The system λ^{Sym}_{Prop}

Definition 1 *1. We have two sets of base types $\mathcal{A} = \{a, b, ...\}$ and $\mathcal{A}^\perp = \{a^\perp, b^\perp, ...\}$.*

2. The set of m-types is defined by the following grammar:

$$A ::= \mathcal{A} \mid \mathcal{A}^\perp \mid A \wedge A \mid A \vee A$$

3. The set of types is defined by the following grammar:

$$C ::= A \mid \perp$$

4. We define the negation $A^\perp$ of an m-type as follows:

- $(a)^\perp = a^\perp$
- $(a^\perp)^\perp = a$
- $(A \wedge B)^\perp = A^\perp \vee B^\perp$
- $(A \vee B)^\perp = A^\perp \wedge B^\perp$

Lemma 2 *For all m-type A, $A^{\perp\perp} = A$.*

Proof: By induction on A. □

Definition 3 *1. The terms of the system λ^{Sym}_{Prop} (called λ_s-terms) are defined (in the natural deduction style) by the following rules:*

$$\frac{}{\Gamma, x : A \vdash x : A}$$

$$\frac{\Gamma \vdash u : A \quad \Gamma \vdash v : B}{\Gamma \vdash \langle u, v\rangle : A \wedge B} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash \sigma_1(t) : A \vee B} \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \sigma_2(t) : A \vee B}$$

$$\frac{\Gamma, x : A \vdash t : \perp}{\Gamma \vdash \lambda x.t : A^\perp} \qquad \frac{\Gamma \vdash u : A^\perp \quad \Gamma \vdash v : A}{\Gamma \vdash u \star v : \perp}$$

We write $\Gamma \vdash_{\lambda_s} t : A$, if we can type the λ_s-term t by the type A using the set of declaration of variables Γ.

2. *The reduction rules are the following:*

$$
\begin{array}{ccll}
(\lambda x.u) \star v & \rightarrow_\beta & u[x := v] & \\
v \star (\lambda x.u) & \rightarrow_{\beta^\perp} & u[x := v] & \\
\lambda x.(u \star x) & \rightarrow_\eta & u & (1) \\
\lambda x.(x \star u) & \rightarrow_{\eta^\perp} & u & (1) \\
\langle u, v\rangle \star \sigma_1(w) & \rightarrow_{\pi_1} & u \star w & \\
\langle u, v\rangle \star \sigma_2(w) & \rightarrow_{\pi_2} & v \star w & \\
\sigma_1(w) \star \langle u, v\rangle & \rightarrow_{\pi_1^\perp} & w \star u & \\
\sigma_2(w) \star \langle u, v\rangle & \rightarrow_{\pi_2^\perp} & w \star v & \\
u[x := v] & \rightarrow_{triv} & v & (2)
\end{array}
$$

(1) *if $x \notin Fv(u)$*

(2) *if u and v are λ_s-terms with type $\perp$, x occurs only one time in u and $u \neq x$. In this case $v = v_1 \star v_2$ and $\lambda y.x$ is a sub-term of u.*

3. *We denote by $\rightarrow$ the one of previous rules. The transitive (resp. reflexive and transitive) closure of $\rightarrow$ is denoted by $\rightarrow^+$ (resp. $\rightarrow^*$).*

4. *We denote the λ_s-terms by small letters like $t, u, v, \ldots$.*

Remark 4 *The reduction $\rightarrow^*$ is not confluent. For example $(\lambda x.(y \star z)) \star (\lambda x'.(y' \star z'))$ reduces both to $y \star z$ and to $y' \star z'$.*

Theorem 5 (Subject reduction) *If $\Gamma \vdash_{\lambda_s} u : A$ and $u \rightarrow^* v$, then $\Gamma \vdash_{\lambda_s} v : A$.*

Proof: It is enough to check that every reduction rule preseves the type. □

Theorem 6 (Strong normalization) *Every λ_s-term is strongly normalizing.*

Proof: See Barbanera and Berardi (1994). □

Remark 7 *Barbanera and Berardi (1994) proved the strong normalization of the λ^{Sym}_{Prop}-calculus by using candidates of reducibility but, unlike the usual construction (for example for Girard's system F), the definition of the interpretation of a type needs a rather complex fix-point operation. This proof is highly non arithmetical. P. Battyanyi recently gave an arithmetical proof of this result by using the methods developed in David and Nour (2005b) to show the strong normalization of systems $\lambda\mu\mu'$- calculus and $\overline{\lambda}\mu\tilde{\mu}$-calculus.*

3 The system CCL

Definition 8 *1. We use the same types as in section 2. The terms of the system* CCL *(called c-terms) are defined (in the Hilbert style) by the following rules:*

$$\overline{\Gamma, x : A \vdash x : A}$$

$$\overline{\Gamma \vdash \mathbf{K} : A^{\perp} \vee (B \vee A)}$$

$$\overline{\Gamma \vdash \mathbf{S} : (A \wedge (B \wedge C^{\perp})) \vee ((A \wedge B^{\perp}) \vee (A^{\perp} \vee C))}$$

$$\overline{\Gamma \vdash \mathbf{C} : (A \wedge B) \vee ((A \wedge B^{\perp}) \vee A^{\perp})}$$

$$\overline{\Gamma \vdash \mathbf{P} : A^{\perp} \vee (B^{\perp} \vee (A \wedge B))}$$

$$\overline{\Gamma \vdash \mathbf{Q_1} : A^{\perp} \vee (A \vee B)} \qquad \overline{\Gamma \vdash \mathbf{Q_2} : B^{\perp} \vee (A \vee B)}$$

$$\frac{\Gamma \vdash U : A^{\perp} \vee B \quad \Gamma \vdash V : A}{\Gamma \vdash (U\ V) : B} \qquad \frac{\Gamma \vdash U : A^{\perp} \quad \Gamma \vdash V : A}{\Gamma \vdash U \star V : \perp}$$

Note that the typed rules does not change the set of declaration of variables. We write $\Gamma \vdash_c T : A$, if we can type the c-term U by the type A using the set a declaration of variables Γ.

2. *Let $U, U_1, U_2, ..., U_n$ be c-terms. We write $(U\ U_1\ U_2\ ...\ U_n)$ instead of $(...((U\ U_1)\ U_2)\ ...\ U_n)$.*

3. *The reduction rules are the following:*

$$\begin{array}{ccc}
(\mathbf{K}\ U\ V) & \triangleright_K & U \\
(\mathbf{S}\ U\ V\ W) & \triangleright_S & ((U\ W)\ (V\ W)) \\
(\mathbf{C}\ U\ V) \star W & \triangleright_{C_r} & (U\ W) \star (V\ W) \\
W \star (\mathbf{C}\ U\ V) & \triangleright_{C_l} & (U\ W) \star (V\ W) \\
(\mathbf{C}\ (\mathbf{K}\ U)\ \mathbf{I}) & \triangleright_{e_r} & U \quad (3) \\
(\mathbf{C}\ \mathbf{I}\ (\mathbf{K}\ U)) & \triangleright_{e_l} & U \quad (3) \\
(\mathbf{P}\ U\ V) \star (\mathbf{Q_1}\ W) & \triangleright_{pq_1} & U \star W \\
(\mathbf{P}\ U\ V) \star (\mathbf{Q_2}\ W) & \triangleright_{pq_2} & V \star W \\
(\mathbf{Q_1}\ W) \star (\mathbf{P}\ U\ V) & \triangleright_{qp_1} & W \star U \\
(\mathbf{Q_2}\ W) \star (\mathbf{P}\ U\ V) & \triangleright_{qp_2} & W \star V \\
W[x := (\mathbf{C}\ (\mathbf{K}\ U)\ (\mathbf{K}\ V))] & \triangleright_{simp} & U \star V \quad (4)
\end{array}$$

(3) *where* $\mathbf{I} = (\mathbf{S}\ \mathbf{K}\ \mathbf{K})$.

(4) *if* W *is a c-term with type* $\perp$.

4. *We denote by $\triangleright$ the one of previous rules. The transitive (resp. reflexive and transitive) closure of $\triangleright$ is denoted by $\triangleright^+$ (resp. $\triangleright^*$).*

5. *We denote the c-terms by capital letters like $T, U, V, \ldots$.*

Remark 9 1. *We have $\vdash_C \mathbf{I} : A^\perp \vee A$ and, for all c-term T, $(\mathbf{I}\ T) \triangleright^* T$.*

2. *The reduction $\triangleright^*$ is not confluent. For example $(\mathbf{C}\ (\mathbf{K}\ y)\ (\mathbf{K}\ z)) \star (\mathbf{C}\ (\mathbf{K}\ y')\ (\mathbf{K}\ z'))$ reduces both to $y \star z$ and to $y' \star z'$.*

Theorem 10 (subject reduction) *If $\Gamma \vdash_c U : A$ and $U \triangleright^* V$, then $\Gamma \vdash_c V : A$.*

Proof: It is enough to check that every reduction rule preserves the type. □

Definition 11 1. *A c-term is said to be pre-term iff it does not contain the symbol $\star$.*

2. *A c-term T is said to be star-term iff $T = U \star V$ for some pre-terms U, V.*

Lemma 12 1. *If A is an m-type and $\Gamma \vdash_c T : A$, then T is a pre-term.*

2. *If $\Gamma \vdash_c T : \perp$, then T is a star-term.*

Proof: Easy. □

Corollary 13 *A c-term is either a pre-term or a star-term.*

Proof: By lemma 12. □

4 The encoding of λ^{Sym}_{Prop} into CCL

Definition 14 *The function $\phi : \lambda^{Sym}_{Prop} \rightarrow \mathrm{CCL}$ is defined as follows:*

- $\phi(x) = x$
- $\phi(\lambda x.t) = l_x(\phi(t))$
- $\phi(u \star v) = \phi(u) \star \phi(v)$
- $\phi(\langle u, v\rangle) = (\mathbf{P}\ \phi(u)\ \phi(v))$
- $\phi(\sigma_1(t)) = (\mathbf{Q_1}\ \phi(t))$
- $\phi(\sigma_2(t)) = (\mathbf{Q_2}\ \phi(t))$

where

- $l_x(x) = \mathbf{I}$
- $l_x(T) = (\mathbf{K}\ T)$ *if T is a pre-term and $x \notin Var(T)$*

- $l_x((U\ V)) = (\mathbf{S}\ \ l_x(U)\ l_x(V))$ *if* $x \in Var((U\ V))$
- $l_x(U \star V) = (\mathbf{C}\ \ l_x(U)\ l_x(V))$

Lemma 15 *Let A and B be m-types.*

1. *If $\Gamma, x : A \vdash_c T : B$, then $\Gamma \vdash_c l_x(T) : A^\perp \vee B$.*
2. *If $\Gamma, x : A \vdash_c T : \perp$, then $\Gamma \vdash_c l_x(T) : A^\perp$.*

Proof: 1. By induction on T.
2. Use 1. □

Theorem 16 *If $\Gamma \vdash_{\lambda_s} t : A$, then $\Gamma \vdash_c \phi(t) : A$.*

Proof: By induction on the typing. Use lemma 15. □

Lemma 17 1. *If U is a pre-term, then $(l_x(U)\ V) \triangleright^* U[x := V]$.*

2. *If U is a star-term, then $l_x(U) \star V \triangleright^* U[x := V]$ and $V \star l_x(U) \triangleright^* U[x := V]$.*

Proof: 1. By induction on U.
2. Use 1. □

Lemma 18 1. *If V is a pre-term and $x \notin Var(V)$, then $l_x(U[y := V]) = l_x(U)[y := V]$.*

2. *$\phi(u[y := v]) = \phi(u)[y := \phi(v)]$.*

Proof: 1. By induction on U.
2. By induction on u. Use 1. □

Remark 19 *As in λ-calculus, we do not have, in general, if $u \to v$, then $\phi(u) \triangleright^+ \phi(v)$. The problem comes from the β-reductions "under a lambda".*

Definition 20 *We write $u \to_\omega v$ if v is obtained by reducing in u a redex which is not within the scope of a λ-abstraction.*

Theorem 21 *If $u \to_\omega v$, then $\phi(u) \triangleright^+ \phi(v)$.*

Proof: By induction on u. Use lemmas 17 and 18. □

5 The encoding of CCL into λ^{Sym}_{Prop}

Notation 22 *Let $\pi_i t$ denote the λ_s-term $\lambda x.(t \star \sigma_i(x))$ where $i \in \{1,2\}$ and $x \notin Fv(t)$. For each $i_1, ..., i_n \in \{1,2\}$, let $\pi_{i_1 \ldots i_n} t$ denote the λ_s-term $\pi_{i_1} ... \pi_{i_n} t$.*

Lemma 23 *1. $\pi_1 \langle u, v\rangle \to^* u$ and $\pi_2 \langle u, v\rangle \to^* v$.*

2. If $\Gamma \vdash_{\lambda_s} t : A \wedge B$, then $\Gamma \vdash_{\lambda_s} \pi_1 t : A$ and $\Gamma \vdash_{\lambda_s} \pi_2 t : B$.

Proof: Easy. □

Notation 24 *Let $[u, v]$ denote the λ_s-term $\lambda x.(u \star \langle v, x\rangle)$ where $x \notin Fv(u) \cup Fv(v)$.*

Lemma 25 *1. $[\lambda x.u, v] \to^* \lambda y.u[x := \langle v, z\rangle]$.*

2. If $\Gamma \vdash_{\lambda_s} u : A^{\perp} \vee B$ and $\Gamma' \vdash_{\lambda_s} v : A$, then $\Gamma, \Gamma' \vdash_{\lambda_s} [u, v] : B$.

Proof: Easy. □

Definition 26 *The function $\psi : \mathrm{CCL} \to \lambda^{Sym}_{Prop}$ is defined as follows:*

- $\psi(x) = x$
- $\psi(\mathbf{K}) = \lambda x.(\pi_1 x \star \pi_{22} x)$
- $\psi(\mathbf{S}) = \lambda x.([[\pi_1 x, \pi_{122} x], [\pi_{12} x, \pi_{122} x]] \star \pi_{222} x)$
- $\psi(\mathbf{C}) = \lambda x.([\pi_1 x, \pi_{22} x] \star [\pi_{12} x, \pi_{22} x])$
- $\psi(\mathbf{P}) = \lambda x.(\langle \pi_1 x, \pi_{12} x\rangle \star \pi_{22} x)$
- $\psi(\mathbf{Q_1}) = \lambda x.(\sigma_1(\pi_1 x) \star \pi_2 x)$
- $\psi(\mathbf{Q_2}) = \lambda x.(\sigma_2(\pi_1 x) \star \pi_2 x)$
- $\psi((U\ V)) = [\psi(U), \psi(V)]$
- $\psi(U \star V) = \psi(U) \star \psi(V)$

Theorem 27 *If $\Gamma \vdash_c U : A$, then $\Gamma \vdash_{\lambda_s} \psi(U) : A$.*

Proof: Use lemmas 23 and 25. □

Lemma 28 $\psi(U[x := V]) = \psi(U)[x := \psi(V)]$.

Proof: By induction on U. □

Theorem 29 *If $U \triangleright V$, then $\psi(U) \to^+ \psi(V)$.*

Proof: The following are easy to check:

$$\begin{array}{ccc}
[[\psi(\mathbf{K}),u],v] & \to^+ & u \\
[[[\psi(\mathbf{S}),u],v],w] & \to^+ & [[u,w],[v,w]] \\
[\psi(\mathbf{I}),u] & \to^+ & u \\
[[\psi(\mathbf{C}),u],v]\star w & \to^+ & [u,w]\star[v,w] \\
w\star[[\mathbf{C},u],v] & \to^+ & [u,w]\star[v,w] \\
[[\psi(\mathbf{C}),[\psi(\mathbf{K}),u]],\psi(\mathbf{I})] & \to^+ & u \\
[[\psi(\mathbf{C}),\psi(\mathbf{I})],[\psi(\mathbf{K}),u]] & \to^+ & u \\
[[\psi(\mathbf{P}),u],v]\star[\psi(\mathbf{Q_1}),w] & \to^+ & u\star w \\
[[\psi(\mathbf{P}),u],v]\star[\psi(\mathbf{Q_2}),w] & \to^+ & v\star w \\
[\psi(\mathbf{Q_1}),w]\star[[\psi(\mathbf{P}),u],v] & \to^+ & w\star u \\
[\psi(\mathbf{Q_2}),w]\star[[\psi(\mathbf{P}),u],v] & \to^+ & w\star v \\
[[\psi(\mathbf{C}),[\psi(\mathbf{K}),u]],[\psi(\mathbf{K}),u]] & \to^+ & \lambda z.(u\star v)
\end{array}$$

For the reduction rule $\triangleright_{simp}$, we use lemma 28. □

Theorem 30 (Strong normalization) *Every c-term is strongly normalizing.*

Proof: By theorems 29 and 6. □

6 Future work

Although the strong normalization of the system CCL follows from the one of the system λ^{Sym}_{Prop} (see theorem 30), R. David and I aim to prove directly this property. We wish to deduce a simpler proof of the strong normalization of the system λ^{Sym}_{Prop}. For that, it is necessary to show a notion stronger than the strong normalization because the coding, presented in section 4, does not simulate all reductions. The verifications we made for the ordinary combinatory logic are very promizing.

In the original combinatory logic the reduction rules of **K** and **S** do not allow β-reduction to be fully simulated (the problem comes from the β-reductions “under a lambda”). Nevertheless, by adding an extensionality rule to combinatory logic (i.e. $\forall x\,\{(F\,x)=(G\,x)\}\Rightarrow F=G$) one obtains an equational theory that corresponds exactly to $\beta\eta$-equivalence. The question is “Is there anything similar for CCL?”. This question is not an easy one because CCL is not confluent. Consequently, a weaker notion than extensionality would be needed.

Acknowledgements

I wish to thank René David for helpful discussions.

References

F. Barbanera and S. Berardi. A symmetric lambda-calculus for classical program extraction. In *TACS'94*, pages 495–515, 1994.

H. Barendregt. *The lambda calculus - Its syntax and semantics*. North Holland, 1984.

A. Church. *The calculi of lambda conversion*. Princeton University Press, 1941.

P. Curien and H. Herbelin. The duality of computation. In *International Conference on Functional Programming*, 2000.

H. Curry and R. Feys. *Combinatory logic, volume 1*. North Holland, 1958.

H. Curry, J. Hindley, and J. Seldin. *Combinatory logic, volume 2*. North-Holland, 1972.

R. David and K. Nour. Why the usual candidates of reducibility do not work for the symetric $\lambda\mu$-calculus. *Electronic Notes in Computer Science*, 140:101–11, 2005a.

R. David and K. Nour. A short proof of the strong normalization of the simply typed $\lambda\mu$-calculus. *Schedae Informaticae*, 12:27–34, 2003a.

R. David and K. Nour. A short proof of the strong normalization of classical natural deduction with disjunction. *Journal of Symbolic Logic*, 68(4):1277 – 1288, 2003b.

R. David and K. Nour. Arithmetical proofs of strong normalization results for symmetric λ-calculi. *Fundamenta Informaticae*, To appear.

R. David and K. Nour. Arithmetical proofs of the strong normalization results for the symmetric $\lambda\mu$-calculus. In *TLCA'05*, pages 162–178, 2005b.

P. DeGroote. A cps-translation of the lambda-mu-calculu. In *CAAP'94*, pages 85–99, 1994.

P. DeGroote. A simple calculus of exception handling. In *TLCA'95*, pages 201–215, 1995.

J.-Y. Girard. A new constructive logic: classical logic. *MSCS*, 1:255–296, 1991.

J. Hindley and J. Seldin. *Introduction to combinators and the lambda calculus*. Cambridge University Press, 1986.

J. Hindley, B. Lercher, and J. Seldin. *Introduction to combinatory logic*. Cambridge University Press, 1972.

M. Holmes. Systems of combinatory logic related to quie's new foundation. *Annals of Pure and Applied Logic*, 53:103–133, 1991.

J.-L. Krivine. Classical logic, storage operators and 2nd order lambda-calculus. *Annals of Pure and Applied Logic*, 68:53–78, 1994.

B. Lercher. Strong reduction and normal forms in combinatory logic. *Journal of symbolic logic*, 32: 213–223, 1967.

M. Parigot. Free deduction: an analysis of computations in classical logic. In *Logic Progr. and Autom. Reasoning*, volume 592, pages 361–380, 1991.

M. Parigot. $\lambda\mu$-calculus: an algorithm interpretation of classical natural deduction. In *LPAR'92*, volume 624, pages 190–201, 1992.

N. Rehof and M. Sorensen. The λ_Δ-calculus. In *TACS'94*, pages 516–542, 1994.

M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924.

P. Wadler. Call-by-value is dual to call-by-name. re-loaded. In *RTA'05*, pages 185–203, 2005.

P. Wadler. Call-by-value is dual to call-by-name. In *International Conference on Functional Programming*, August 2003.