Discharging sub-derivations: A proof-theoretic Curry-Howard correspondence for a λ-calculus with patterns

Nissim Francez
Computer Science dept.,
Technion-IIT, Haifa, Israel
francez@cs.technion.ac.il

January 23, 2008

Abstract From the perspective of the Curry-Howard correspondence (CH), *abstraction*

over a variable in the simply-typed λ-calculus λ-calculus TA_λ [3]corresponds to the proof-theoretic notion of *discharge of an assumption* in implication-introduction within natural-deduction (ND) proof-systems. Various (propositional) logics and λ-calculi can be obtained by side-conditions on the discharge/abstraction (see, e.g., [2] for side-conditions on the number of occurrences of the free abstracted variable in the body of the expression). Similarly, application in the λ-calculus corresponds by CH to implication-elimination in ND.

In [4], a distinction is made between *specific* and *non-specific* assumptions. The former are introduced in an ND proof/derivation "according to their meanings", while the latter are mere "placeholders" (for a closed proof). While [4] focuses on differences in *introducing* the two kinds of assumptions, the current paper focuses on differences in the *discharge* of the two kinds.

In the standard (simply-typed) λ-calculus, abstraction is based on non-specific assumptions. The term $\lambda x.M$ does not impose any restrictions on the "role" of x within M. This view is corroborated by the standard β-reduction, that allows the (capture fee) substitution of an *arbitrary* term N for x in M, as the result of applying $\lambda x.M$ to N.

In this paper, I study a generalization of the λ-calculus that *does allow* imposing restrictions on the form of N as a precondition to a reduction step. This results from *specific abstraction* over assumptions producing a term, with which N has to *match* (defined in the paper) for the reduction to take place. Formally, a new term is introduced, having the form $\lambda[M'].M$, where M' is itself a term, not necessarily a variable. This term is interpreted proof-theoretically as a *discharge of a sub-derivation* (with all its open assumptions). By identifying $\lambda[x].M$ with $\lambda x.M$ we recover the usual λ-calculus as a sub-calculus.

A calculus like this, called the $\lambda\phi$-calculus, was presented in [5], with the aim of *pattern matching* in functional programming. The focus there is is on establishing *confluence* of the induced reduction. Later, [1] studied typing in such functional language. None of them considers the proof-theoretical interpretation of the calculus, in particular not sub-proof discharge.

Technically, the proposed generalization amounts to an *interdependent simultaneous abstraction* of a collection of related variables: when well-typed, $\lambda[M'].M$ binds simultaneously the *free* variables of M', imposing a mutual-dependence relation over them. The role of the bound variables in M' is an auxiliary means for expressing the required dependency relation, as is explained in the paper.

The system $TA_{\hat{\lambda}}$ is presented below.

$$\frac{[\Gamma_1]_i \cup \Gamma_2 \vdash Q : \tau \quad \left[\ \Gamma_1 \vdash P : \sigma\ \right]_i}{\Gamma_2 \vdash \lambda[P].Q : (\sigma \to \tau)} (\to I_i) \quad \frac{\Gamma_1 \vdash \lambda[P].Q : \sigma \to \tau \quad \Gamma_2 \vdash P' : \sigma}{\Gamma_1 \Gamma_2 \vdash sQ : \tau} (\to E), \quad \text{where } s = \mu(P, P')$$

Here s is the substitution produced by matching P and P'. The second premise is called a *licensing derivation.* Below are two generalized typing examples. Here A, B, C range over proposition variables, and σ, τ range over wffs in the implicational fragment of the propositional calculus.

Abstracting application: Consider the (simply-typed) identity function $\lambda x.x : B \to B$. Suppose we want to restrict it to terms of type B that are applications (uv) (hence, u is of type $A \to B$, and v of type A, for some A and B). This is achieved by abstracting *simultaneously* two assumptions, which establishes the type B for (uv), thereby recording in the term that this type was formed by an application, abstracted over.

$$\frac{\dfrac{[u : A \to B]_1 \quad [v : A]_1}{(uv) : B} (\to E) \quad \left[\dfrac{u : A \to B \quad v : A}{(uv) : B} (\to E) \right]_1}{\lambda[(uv)].(uv) : B \to B} (\to I_1)$$

First, we observe that the conclusion is of the required form. It is of the type of the identity function, and its term restricts application (via $\hat{\beta}$-reduction) only to terms in the form of an application. What we did is to abstract simultaneously over $\{u, v\}$ (by discharging simultaneously a "package" of two different assumptions, indexed 1 in the example), producing a type (B in the example), that becomes an antecedent of an implication based on an auxiliary derivation from the discharged assumptions, to be called a *licensing derivation.* The licensing (sub-)derivation is discharged together with the discharged assumptions.

Consider another example, showing the effect on context (undischarged assumptions).

$$\frac{\dfrac{[u : (A \to B)]_1 \quad \dfrac{x : (A \to A) \quad [v : A]_1}{(xv) : A} (\to E)}{(u(xv)) : B} (\to E) \quad \left[\dfrac{u : (A \to B) \quad v : A}{(uv) : B} (\to E) \right]_1}{\lambda[(uv)].(u(xv)) : (B \to B)} (\to I_1)$$

Here x has to be of type $A \to A$ for the term to be well-typed.

Abstracting abstraction: Consider next the term $\lambda[\lambda w.(u(wv))].(u(xv))$. Here too, the abstracted term will have to be well-typed, restricting simultaneously both u and v, *but not* w, abstracted over internally (reflecting an assumption discharge within the licensing derivation). The restriction is that u must have a type applicable to a function applied to (xv), not to v itself. Thus, in the body of the generalized term, (*the free!*) x will reflect an assumption having such a type.

$$\frac{\dfrac{[u : (B \to C)]_1 \quad \dfrac{x : (A \to B) \quad [v : A]_1}{(xv) : B} (\to E)}{(u(xv)) : C} (\to E) \quad \left[\dfrac{\dfrac{u : (B \to C) \quad \dfrac{[w : (A \to B)]_2 \quad v : A}{(wv) : B} (\to E)}{(u(wv)) : C} (\to E)}{\lambda w.(u(wv)) : ((A \to B) \to C)} (\to I_2) \right]_1}{\lambda[\lambda w.(u(wv))].(u(xv)) : (((A \to B) \to C) \to C)} (\to I_1)$$

References

[1] Gilles Barthe, Horatiu Cirstea, Claude Kirshner, and Luigi Liquori. Pure patterns type systems. In *Proceedings of Principles of Programming Languages (POPL'03)*, pages 250–272. ACM, January 2003.

[2] Dov Gabbay and Ruy J. G. B. De Queiroz. Extending the curry-howard interpretation to linear, relevant and other resource logics. *J. of Symbolic Logic*, 57(4):1319–1365, December 1992.

[3] J. Roger Hindley. *Basic simple Type Theory.* Cambridge University Press, 1997.

[4] Peter Schroeder-Heister. On the notion of *assumption* in logical systems. In R. Bluhm and C. Nimtz, editors, *Philosophy-Science-Scientific Philosophy.* Mentis, Paderborn, 2004. Selected papers of the 5th int. congress of the society for Analytic Philosophy, Bielfield, September 2003.

[5] Vincent van Oostrom. Lambda calculus with patterns. Technical Report IR-228, Vrije Universiteit, Amsterdam, 1990.