Strong normalization by reducibility of the weak λ_P calculus

Julien FOREST
LRI (CNRS UMR 8623)
Bât 490, Université Paris-Sud
91405 Orsay Cedex
France
e-mail:forest@lri.fr

Pattern calculi [2, 3] were proposed as a theoretical modelization of programming languages with function definitions by cases using pattern matching such as CAML [1], Haskell [6], ML [7], etc.
In this paper we study strong normalization of a new pattern calculus called λ_P. The main differences between terms of λ_P and those of classical λ-calculus are:

- The construction $\lambda x.M$ of the λ-calculus is replaced by $\lambda P.M$ where P is a pattern which belongs to a given grammar including a notion of choice between two different patterns to denote patterns of sum types. Thus for example one can write terms such as $A = \lambda(nil \mid_\xi cons(h,t)).[nil|_\xi t]$ which denotes a function which takes a list and returns another list: when the argument is an empty list nil then the function A returns an empty list nil, and when A is applied to a non-empty list $cons(h,t)$, it returns the tail t of this list $cons(h,t)$.
- A set of variables (called choice variables) dealing with the notion of choice between two patterns is added. For example, the variable ξ of the previous example A belongs to this new set.

The main differences between the pattern calculi in [2, 3] and λ_P are:

- λ_P treats explicitly both pattern matching and substitutions (including substitutions generated by the choice of patterns).
- Our grammar is much more in the classical lambda calculus style.
- A system of substitution with concatenation (such as the one used in λ_σ [4]) is introduced to preserve confluence.

The calculus λ_P is weak (i.e. substitutions can not cross lambdas [5]) and is a typed calculus which verifies subject reduction and confluence.
The λ_P calculus can be seen as a generalization of the well-known weak λ_σ calculus (but in named notation) denoted here $\lambda_{\sigma w}$. For this reason we first

give a proof of strong normalization for $\lambda_{\sigma w}$, which is then extended to deal with patterns, thus obtaining a proof of strong normalization for λ_P.
The proof for $\lambda_{\sigma w}$ is inspired by the proof of termination of Ritter [8] which is based on the reducibility technique and which can be summarized as follows:
We first define a congruence $\equiv$ on the $\lambda_{\sigma w}$ term. We then define a reduction relation, denoted $\rightarrow_{\equiv}$, on the quotient of the set of terms. We then prove that the relation $\rightarrow_{\equiv}$ is strong normalizing as follows:

1. We define reducible terms and reducible substitutions for a given environment.

2. We prove that if a term or a substitution is reducible in an environment then it is $\rightarrow_{\equiv}$-strong normalizing.

3. We prove that every term or substitution which is typable in an environment is also reducible.

Finally, we show by a technical Lemma that strong normalization of $\rightarrow_{\equiv}$ implies strong normalization of $\lambda_{\sigma w}$.
We extend the proof of strong normalization for $\lambda_{\sigma w}$ to λ_P using the same ideas. We redefine the previous notions in the new context of both pattern and multiple sorts of variables (usual and choice variables). This proof is much more technical that the $\lambda_{\sigma w}$ but follows the same scheme than the one for $\lambda_{\sigma w}$.

References

[1] The Objective Caml language. `http://caml.inria.fr/`.

[2] Val Breazu-Tannen, Delia Kesner, and Laurence Puel. A typed pattern calculus. In *Proceedings of the Eigth Annual IEEE Symposium on Logic in Computer Science*, pages 262–274. IEEE Comp. Soc. Press, 1993.

[3] Serenella Cerrito and Delia Kesner. Pattern matching as cut elimination. In Giuseppe Longo, editor, *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, Trento, Italy, July 1999. IEEE Comp. Soc. Press.

[4] Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, March 1996.

[5] Therese Hardin, Luc Maranget, and Bruno Pagano. Functional back-ends within the lambda-sigma calculus. Technical Report RR-3034, Inria, Institut National de Recherche en Informatique et en Automatique.

[6] P. Hudak, S. Peyton-Jones, and P. Wadler (editors). Report on the programming language haskell, a non-strict, purely functional language (version 1.2). Sigplan Notices, 1992.

[7] Lawrence C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1991.

[8] E. Ritter. Normalisation for typed lambda calculi with explicit substitution. *Lecture Notes in Computer Science*, 832:295–??, 1994.