Encoding rewriting strategies in lambda-calculi with patterns

Germain Faure

ENCODING REWRITING STRATEGIES IN λ-CALCULI WITH PATTERNS

by

Germain Faure

Abstract. — We propose a patch to the pure pattern calculus: we claim that this is strictly more powerful to define the application of the match fail as the pure λ-term defining the boolean `false` instead of the identity function as it is done in the original version of the pure pattern calculus [**JK09**].

We show that using non algebraic patterns we are able to encode in a natural way any rewriting strategies as well as the branching construct | used in functional programming languages.

We close the open question (raised in [**Cir00, CK01**]) whether rewriting strategies can be directly encoded in λ-calculi with patterns.

1. Preliminaries

We first recall some classical encoding of pairs, boolean etc. in the pure λ-calculus (see for example [**Bar84**])

$$\begin{array}{lcl}
\langle t,u\rangle & = & \lambda x.\, x\ t\ u \\
\pi_1 t & = & t\ (\lambda xy.\, x) \\
\pi_2 t & = & t\ (\lambda xy.\, y) \\
 & & \\
\texttt{true} & = & \lambda xy.\, x \\
\texttt{false} & = & \lambda xy.\, y \\
\texttt{if}\ t\ \texttt{then}\ u\ \texttt{else}\ v & = & t\ u\ v \\
 & & \\
\texttt{let}\ x = t\ \texttt{in}\ u & = & (\lambda x.\, u)\ t \\
 & & \\
\texttt{fix} & = & (\lambda x.\, \lambda f.\, f(x\ x\ f))\ (\lambda x.\, \lambda f.\, f(x\ x\ f))
\end{array}$$

If $C[]$ is a context then we will write $fun = \texttt{fix}(C[fun])$ for the definition of a recursive function called fun and defined by $\texttt{fix}\ (\lambda s.\, C[s])$.

2. Yet a more general framework

We consider the general framework given in Section 2 of [**JK09**] and use the same notations. We first begin by a remark. The application of a match μ to a term can be defined in a more general way as follows: If μ is a substitution, then the application of the match to a term is obtained by applying the substitution to variables of the term as explained in [**JK09**]. If μ is `fail`, we define

$$\texttt{fail}\ t = u$$

where u is an arbitrary term [the calculus is parametrized by this term u].

***Theorem 2.1* (Confluence).** — *The pure pattern calculus, as defined in Section 3 of* [**JK09**] *but with the above application of a match, is confluent when u is a pure λ-term in normal form.*

Proof. — Exactly the same as the one of the original paper. Note that the case

$$\texttt{fail}\ t = [x]\ \widehat{x} \to x$$

is now subsumed. □

3. Encoding strategies

In this section, we give a semantics to rewriting strategies as they are used in rewriting-based languages such as Tom [**BBK$^+$07**] or Stratego [**Vis01**]. We refer for example to [**Rei06, BMR08**] for their presentation.

In the original paper on the rewriting calculus, rewriting strategies were encoded in an ad-hoc extension, using what they called the `first` operator. But the question whether rewriting strategies can be directly encoded in λ-calculi with patterns were remained open.

3.1. Informal presentation of the encoding. — To encode strategies, we need to test failure and success, in particular to encode the choice operator. In the following, a strategy is going to be encoded by a function returning a pair made of first a boolean whose value depends on the success or failure of the strategy application and secondly, in case of success, the second term of the pair represent the result of the strategy application. We then use the following encoding[(1)]

$$\texttt{first}(t_1, t_2) \quad = \quad \lambda x.\texttt{let}\ a = t_1\ x\ \texttt{in if}\ \pi_1 a\ \texttt{then}\ a\ \texttt{else}\ t_2\ x$$

3.2. Definition and properties of the encoding. — We now suppose given an instance of the pure pattern calculus where the application of the match `fail` is given by:

$$\texttt{fail}\ t = \texttt{false}$$

where `false` is defined in Section 1.

We want that the function ϕ giving the encoding of strategies in the pure pattern calculus (and defined below) satisfies the following theorem

Theorem 3.1. — *For any strategy s, for any terms t and u: if $t \to_s^* u$ (t rewrites to u under the strategy s) then*

$$\phi(s)\ t \to^* \begin{cases} \langle \texttt{false}, \_ \rangle & \text{if } u \text{ a failure} \\ \langle \texttt{true}, u \rangle & \text{otherwise} \end{cases}$$

where $\_$ denotes a term depending on s, t, u and of which we don't care the value.

Definition 3.2. — The function ϕ is inductively defined in Fig. 1.

As usual, in the encoding of $one(s)$, the order of application of s to the children is fixed. It is easy to prove that Th. 3.1 holds for the definition of ϕ given in Fig. 1.

Remark 3.3. — Some other strategies can be encoded either using the above operators:

$$\begin{array}{lcl} choice(s_1, s_2) & = & \texttt{IfThenElse}(s_1,s_1,s_2) \\ try(s) & = & \texttt{IfThenElse}(s,s,\texttt{Identity}) \\ not(s) & = & \texttt{IfThenElse}(s,\texttt{Fail},\texttt{Identity}) \end{array}$$

or directly (which is slightly more efficient for *choice* and *try*):

$$\begin{array}{lcl} \phi(choice(s_1, s_2)) & = & \lambda x.\texttt{let}\ a = \phi(s_1)\ x\ \texttt{in}\ (\texttt{if}\ \pi_1\ a\ \texttt{then}\ a\ \texttt{else}\ \phi(s_2)\ x) \\ \phi(try(s)) & = & \lambda x.\texttt{let}\ a = \phi(s)\ x\ \texttt{in}\ (\texttt{if}\ \pi_1\ a\ \texttt{then}\ a\ \texttt{else}\ x) \end{array}$$

(1) recalling the `first` operator of [**Cir00, CK01**]

$$
\begin{array}{lcl}
\phi(\texttt{Identity}) & = & \lambda x.<\texttt{true}, x> \\
\phi(\texttt{Fail}) & = & \lambda x.<\texttt{false}, \_> \\
\phi(l \to r) & = & \lambda x.\langle (l \to \texttt{true})\ x,\ (l \to r)\ x\rangle \\
\phi(s_1; s_2) & = & \lambda x.\texttt{let}\ a = \phi(s_1)\ x\ \texttt{in}\ (\texttt{if}\ \pi_1\ a\ \texttt{then}\ \phi(s_2)\ a\ \texttt{else}\ <\texttt{false}, \_>) \\
\phi(\texttt{IfThenElse}(s_1,s_2,s_3)) & = & \lambda x.\texttt{let}\ a = \phi(s_1)\ x\ \texttt{in}\ (\texttt{if}\ \pi_1\ a\ \texttt{then}\ \phi(s_2)\ x\ \texttt{else}\ \phi(s_3)\ x) \\
\phi(all(s)) & = & \texttt{fix}(\lambda x.\texttt{first}(\\
 & & \quad x_1x_2 \to \texttt{let}\ a = \phi(s)\ x_2\ \texttt{in} \\
 & & \qquad\qquad \texttt{if}\ (\pi_1 a)\ \texttt{then let}\ b = \phi(all(s))\ x_1\ \texttt{in}\ \langle \pi_1 b,\ (\pi_2 b)\ a\rangle \\
 & & \qquad\qquad\qquad \texttt{else}\ \langle \texttt{false}, \_\rangle, \\
 & & \quad y \to \langle \texttt{true}, y\rangle) \\
 & & \qquad\qquad x) \\
\phi(one(s)) & = & \texttt{fix}(\lambda x.\texttt{first}(\\
 & & \quad x_1x_2 \to \texttt{let}\ a = \phi(s)\ x_2\ \texttt{in} \\
 & & \qquad\qquad \texttt{if}\ \pi_1 a\ \texttt{then}\ \langle \texttt{true},\ x_1\ a\rangle \\
 & & \qquad\qquad\qquad \texttt{else let}\ b = \phi(one(s))\ x_1\ \texttt{in}\ \langle \pi_1 b,\ (\pi_2 b)\ x_2\rangle \\
 & & \quad y \to \langle \texttt{false}, y\rangle) \\
 & & \qquad\qquad x)
\end{array}
$$

FIGURE 1. Encoding of rewriting strategies

Remark 3.4. — **Encoding of the branching construct** To define the branching construct we can use the *choice* operator. But, in the case of pattern-matching defined in functional programming languages, pattern-matching is exhaustive and then the "final" result cannot be a failure. We thus define

$$t_1|\dots|t_n = \lambda x.\,\pi_2\Big(\phi(choice(t_1,\dots,t_n))\ x\Big)$$

Acknowledgements. — H. Cirstea, D. Kesner and B. Valiron for discussions on the subject of this work.

References

[Bar84] H. BARENDREGT – *The Lambda-Calculus, its syntax and semantics*, Studies in Logic and the Foundation of Mathematics, North Holland, 1984, Second edition.

[BBK+07] E. BALLAND, P. BRAUNER, R. KOPETZ, P.-E. MOREAU & A. REILLES – "Tom: Piggybacking rewriting on java", *Proceedings of the 18th Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, vol. 4533, 2007, p. 36–47.

[BMR08] E. BALLAND, P.-E. MOREAU & A. REILLES – "Rewriting strategies in java", *Electronic Notes in Theoretical Computer Science* **219** (2008), p. 97–111.

[Cir00] H. CIRSTEA – "Calcul de réécriture : fondements et applications", Thèse de Doctorat, Université Henri Poincaré - Nancy I, 2000.

[CK01] H. CIRSTEA & C. KIRCHNER – "The rewriting calculus — Part I *and* II", *Logic Journal of the Interest Group in Pure and Applied Logics* **9** (2001), no. 3, p. 427–498.

[JK09] B. JAY & D. KESNER – "First-class patterns", (2009).

[Rei06] A. REILLES – "Réécriture et compilation de confiance", Ph.D. Thesis, November 2006.

[Vis01] E. VISSER – "Stratego: A language for program transformation based on rewriting strategies", *Rewriting Techniques and Applications - RTA'01*, Lecture Notes in Computer Science, vol. 2051, 2001.

September 3, 2009

GERMAIN FAURE