Guarded Exception Handling: some results (abstract)

Catherine Pilière

UHP-LORIA
BP 239
54 506 Vandœuvre-lès-Nancy Cedex France
e-mail: Catherine.Piliere@loria.fr

1 Introduction

In [1] Philippe de Groote proposed a computational interpretation of classical logic through a simply typed λ-calculus featuring an exceptions handling mechanism inspired by the ML language. The most interesting property of the calculus is given by its typing system which ensures that every raised exception is eventually handled whenever the whole term is correctly typed. This result is achieved by means of *static* exception handlers which act as safeguards for each locally declared exception, by preventing them to escape their scope of declaration. From a theoretical point of view, this is due to the fact that the corresponding reduction rule satisfies the subject reduction property.

The problem is that *static* exceptions handling mechanism is a strong restriction that obviously does not offer the full power of the classical, "real" exceptions handling it à la ML [4]. In particular, Lillibridge's result [2] becomes no more available since static exceptions handling appears to be equivalent to Scheme's call with current continuation (*call/cc*). Nevertheless, Turing-completeness can anyway be retained in a more natural way by adding a general fixed-point operator to the calculus. The most serious reservation thus concerns the practical use of static exception handlers.

We here show how dynamic handlers can be added to the calculus in a realistic programming context authorizing general recursion, while preserving the interesting property that well-typed programs cannot evaluate to uncaught exceptions.

2 Exceptions declarations and static handlers

Definition 1. The types of the calculus are given by the following grammar: $\mathcal{T} \ ::= \ \iota \mid exn \mid \mathcal{T} \rightarrow \mathcal{T}$ where ι and exn stand for the base types, exn being the type of exceptional values.

λ_{exn} features an exception handling mechanism by means of exception variables y which act as datatype constructors. They are first-class values of functional type (e.g. $\tau \rightarrow exn$, which then return exceptions when applied to a term of the corresponding type (e.g. τ). As first-class citizens, exceptions may be passed as parameters and return as values. Moreover, any exception $y\,V$ may also be raised under the form of packets $\mathcal{R}(y\,V)$, which are then propagated and possibly handled.

The key point of λ_{exn} is that exceptions constructors are locally declared together with an associated *static* handler. Unlike dynamic handlers à la ML, static handlers are kept during the evaluation, thus ensuring that the exceptions they guarded will not escape their scope of declaration. In the following, we let $\langle y.\, M \mid x.\, N \rangle$ stand for the local exception declaration $let\ y : \alpha \rightarrow exn\ in\ M\ handle\ (yx) \Rightarrow N$. We also allow recursive definitions of functions by means of a general recursion operator μ.

Definition 2. The terms of the calculus are the following:

$$\mathcal{T} \ ::= \ c_i \mid x \mid y \mid \lambda x.\, \mathcal{T} \mid (\mathcal{T}\mathcal{T}) \mid (\texttt{raise}\ \mathcal{T}) \mid \langle y.\, \mathcal{T} \mid x.\, \mathcal{T} \rangle \mid \mu f.\, \lambda x.\, \mathcal{T},$$

where $\{c_i \mid i \in \omega\}$ is a set of ground type constants.

The terms of the calculus evaluate under the call-by-value mechanism. We thus have to define the notion of *value*:

Definition 3. The set $\mathcal{V}$ of syntactic values is defined by: $\mathcal{V} \ ::= \ c_i \mid x \mid y \mid \lambda x.\, M \mid (y\mathcal{V})$.

Definition 4. Let Γ stand for a *typing environment*, *i.e.*, a partial function that assigns types to variables. The terms are given a type wrt the following rules:

$$\Gamma \vdash c_i : \iota \qquad \Gamma \vdash x : \Gamma(x) \qquad \Gamma \vdash y : \Gamma(y)$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x. M : \alpha \to \beta} \qquad \frac{\Gamma, x : \alpha, f : \alpha \to \alpha \vdash M : \alpha}{\Gamma \vdash \mu f. \lambda x. M : \alpha \to \alpha}$$

$$\frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta} \qquad \frac{\Gamma \vdash M : exn}{\Gamma \vdash \mathcal{R}(M) : \alpha}$$

$$\frac{\Gamma, y : \alpha \to exn \vdash M : \beta \quad \Gamma, x : \alpha \vdash N : \beta}{\Gamma \vdash \langle\, y. M \mid x. N \,\rangle : \beta}$$

We also define the two following notions of *contexts* (i.e., terms with one "hole"):

Definition 5. The notion of *exception declaration context* is defined by

$$\mathcal{D} ::= [\,] \mid \langle\, y. \mathcal{D} \mid x. \mathcal{T} \,\rangle$$

Definition 6. The evaluation proceeds in the applicative order. We thus have the following notion of *evaluation context*

$$\mathcal{E} ::= [\,] \mid (\mathcal{V}\,\mathcal{E}) \mid (\mathcal{E}\,\mathcal{T}) \mid \mathcal{R}\,(\mathcal{E})$$

Definition 7. Let $\mathscr{D}$ stand for a $\mathcal{D}$-context. The one-step reduction relation of the calculus is the least relation compatible with the terms formation rules and containing the following notions of reduction:

$$\begin{aligned}
(\lambda x. M)\, V &\to_{\lambda_{exn}} M[x := V] && (1)\\
\mu f. \lambda x. M &\to_{\lambda_{exn}} \lambda x. M[f := \mu f. \lambda x. M] && (2)\\
\langle\, y. \mathscr{D}\,[\mathcal{R}\,(y\, V)] \mid x. H \,\rangle &\to_{\lambda_{exn}} \langle\, y. \mathscr{D}\,[H[x := V]] \mid x. H \,\rangle && (3)\\
V\, \mathcal{R}\,(M) &\to_{\lambda_{exn}} \mathcal{R}\,(M) && (4)\\
\mathcal{R}\,(M)\, N &\to_{\lambda_{exn}} \mathcal{R}\,(M) && (5)\\
\mathcal{R}\,(\mathcal{R}\,(M)) &\to_{\lambda_{exn}} \mathcal{R}\,(M) && (6)\\
V\,\langle\, y. M \mid x. H \,\rangle &\to_{\lambda_{exn}} \langle\, y. V\, M \mid x. V\, H \,\rangle && (7)\\
\langle\, y. M \mid x. H \,\rangle\, O &\to_{\lambda_{exn}} \langle\, y. M\, O \mid x. H\, O \,\rangle && (8)\\
\mathcal{R}\,(\langle\, y. M \mid x. H \,\rangle) &\to_{\lambda_{exn}} \langle\, y. \mathcal{R}\,(M) \mid x. \mathcal{R}\,(H) \,\rangle && (9)\\
\langle\, y. M \mid x. H \,\rangle &\to_{\lambda_{exn}} M \quad \text{if } y \notin FV(M) && (10)
\end{aligned}$$

The system is confluent [5] and satisfies the subject reduction property (by induction of the derivation of $M \to_{\lambda_{exn}} N$).

Definition 8. The one-step evaluation relation between λ_{exn}-terms is defined by the following computation rules:

$$\begin{aligned}
\mathscr{D}\,[\mathscr{E}[(\lambda z. M)\, V]] &\xrightarrow{\text{eval}} \mathscr{D}\,[\mathscr{E}[M[z := V]]]\\
\mathscr{D}\,[\mathscr{E}[\mu f. \lambda x. M]] &\xrightarrow{\text{eval}} \mathscr{D}\,[\mathscr{E}[\lambda x. M[f := \mu f. \lambda x. M]]]\\
\mathscr{D}_1[\langle\, y. \mathscr{D}_2[\mathscr{E}[\mathcal{R}\,(y\, V)]] \mid x. M \,\rangle] &\xrightarrow{\text{eval}} \mathscr{D}_1[\langle\, y. \mathscr{D}_2[M[x := V]] \mid x. M \,\rangle]\\
\mathscr{D}\,[\mathscr{E}[\langle\, y. M \mid x. N \,\rangle]] &\xrightarrow{\text{eval}} \mathscr{D}\,[\langle\, y. \mathscr{E}[M] \mid x. \mathscr{E}[N] \,\rangle] \quad (\mathscr{E} \neq [\,])
\end{aligned}$$

By means of a continuation passing style denotational semantics that we proved to be computationally adequate, we can show that the transitive, reflexive closure of the evaluation relation above defines a normalizing strategy for the programs (*i.e.*, the closed terms of ground type ι). It follows that well typed programs' evaluation gives not rise to uncaught exception neither replaces possible escaping exceptions with non-termination. This establishes that the use of static exceptions handlers confines any exception to the scope of its declaration in a sound way [1] .

[1] These results have been obtained in collaboration with Philippe de Groote. They are not yet published

3 Adding dynamic handlers

Now, provided that they appear *inside* the guarded scope of a $\mathcal{D}$-context, it is then possible to allow the presence of dynamic handlers *à la ML* for any locally declared exception.

Definition 9. The notion of *dynamic handlers* extends the terms' grammar with the following item:

$$\mathcal{T}\{yx \Rightarrow \mathcal{T}\}.$$

A dynamic handler is a *match*, not a term. The set of values is then kept unchanged.

A term followed with a dynamic handler is typed thanks to the new following rule:

Definition 10. Let Γ stand for a typing environment. The typing rule for dynamic handlers is given by:

$$\frac{\Gamma,\, y : \alpha \to exn \;\vdash\; M : \beta \quad \Gamma,\, y : \alpha \to exn,\, x : \alpha \;\vdash\; N : \beta}{\Gamma,\, y : \alpha \to exn \;\vdash\; M\{yx \Rightarrow N\} : \beta}$$

Dynamic handlers define a third notion of context:

Definition 11. An *exception handling context* is described by the following grammar

$$\mathcal{H} ::= [\,] \mid \mathcal{H}\,\{(y\,x) \Rightarrow \mathcal{T}\}$$

The way dynamic handlers work is quite similar to the rule for static exceptions handlers [3]. They only differ in that dynamic handlers are not kept whereas static handlers are. We propose to enrich the calculus of section 2 with the three following notions of reductions:

Definition 12.

$$\mathcal{R}\,(y\,V)\,\{(y\,x) \Rightarrow H\} \to_{\lambda_{exn}} H[x := V] \tag{11}$$
$$\mathcal{R}\,(y\,V)\,\{(z\,x) \Rightarrow H\} \to_{\lambda_{exn}} \mathcal{R}\,(y\,V) \tag{12}$$
$$V\,\{(y\,x) \Rightarrow H\} \to_{\lambda_{exn}} V \tag{13}$$

In rule 11, the dynamic handler matches the raised exception: the handler is invoked. Rule 12 expresses what happens when a dynamic handler does not match the raised exception: the handler simply passes the packet on. In rule 13, the handler is useless since no packet has been raised.

It is important to note that in any case, unlike static handlers in rule 3, dynamic handlers are lost. However, all the variables and exceptional variables which occur bound in the left hand side of each rule above still remain bound in the right hand side. It is then possible to prove that adding these rules preserve the subject reduction of the calculus which is the key property used in [1] to show that well typed closed terms of type ι cannot evaluate to raised uncaught exceptions.

We can now express the way programs including dynamic handlers evaluate wrt the previously defined reduction rules ($\mathscr{H} \in \mathcal{H}$):

Definition 13.

$$\mathscr{D}\,[\mathscr{H}[\mathscr{E}\,[\mathcal{R}\,(y\,V)]\,\{(y\,x) \Rightarrow N\}]] \xrightarrow{\text{eval}} \mathscr{D}\,[\mathscr{D}\,[N[x := V]]]$$
$$\mathscr{D}\,[\mathscr{H}[[\mathscr{E}\,[\mathcal{R}\,(y\,V)]]\,\{(z\,x) \Rightarrow N\}]] \xrightarrow{\text{eval}} \mathscr{D}\,[\mathscr{H}[\mathcal{R}\,(y\,V)]]$$
$$\mathscr{D}\,[\mathscr{D}\,[\mathscr{E}\,[V\,\{(y\,x) \Rightarrow N\}]]] \xrightarrow{\text{eval}} \mathscr{D}\,[\mathscr{D}\,[\mathscr{E}\,[V]]]$$

4 Conclusions

Definition 8 completed with Definition 13 [2] provide a notion of *guarded exception handling* which prevents locally declared exceptions to escape their scope of declaration in a way that seems to be sound, even in the presence of general recursion. It then could be interesting to study the paradigm in practical use through an abstract machine akin to the SECD machine of Landin.

[2] of course, exceptions handling contexts $\mathscr{H}$ must be added in the rules ofDefinition 8 in the same way, but this is immediate and then left to the reader due to lack of space

References

1. Ph. de Groote. A simple calculus of exception handling. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Second International Conference on Typed Lambda Calculi and Applications, TLCA'95*, volume 902 of *Lecture Notes in Computer Science*, pages 201–215. Springer Verlag, 1995.
2. M. Lillibridge. Exceptions are strictly more powerful that call/cc. Technical Report CMU-CS-95-178, Carnegie Mellon University, 1995.
3. R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. The MIT Press, 1990.
4. L.C. Paulson. *ML for the working programmer*. Cambridge University Press, 1991.
5. C. Pilière. A confluence result for a typed lambda-calculus of exceptions handling with fixed-point. In G. Ciobanu and G. Păun, editors, *12th International Symposium on Fundamentals of Computation Theory*, volume 1684 of *Lecture Notes in Computer Science*, pages 421–432. Springer Verlag, 1999.