Designing interrupts for ML and OCaml

Guillaume Munch-Maccagnoni, Leo White, Stephen Dolan

Designing interrupts for ML and OCaml

Guillaume Munch-Maccagnoni*
Leo White†
Stephen Dolan‡

14th June 2023

This talk reports on ongoing work to propose a sound design for asynchronous exceptions (or *interrupts*, in ML parlance) in the OCaml language, inspired by fault-tolerance concepts and Rust's design for exceptions.

We first describe the problems of recovering from interrupts in Standard ML and OCaml, and how these language deficiencies have been worked around, notably in the implementation of proof assistants and other interactive tools.

We then aim to explain more conceptually how safe interrupt handling is possible by evolving ML exceptions. We emphasize key notions: exception-safety concepts originating from C++ and Rust's extension of Erlang's "let it fail" approach to shared mutable state. Lastly we will discuss considerations specific to the OCaml language.

This work suggests, in the spirit of structured programming, that while the ML notion of exception lacks structure, it refines into better-structured abstractions for error handling, suggested by its relevant usage by programmers.

Interrupts in ML ensure the prompt termination of the program due to external events, such as the user pressing Ctrl-C on the keyboard. We first review the usage of interrupts in ML and OCaml. We first underline its use in proof assistants and other interactive tools, based on a survey of existing OCaml code published on the public opam repository. We also point out its use in Multicore OCaml due to the need for task cancellation.

Interrupts in ML and OCaml are implemented as asynchronous exceptions: exceptions that can arise out of thin air at an arbitrary point in the program. This design comes with considerable challenges, for instance to reason about the state of the program when an interrupt is caught, and thus the ability to recover and proceed with further user interaction. This challenge is exacerbated in application domains closer to systems programming—an application area which

*guillaume.munch-maccagnoni@inria.fr, INRIA, France
†lwhite@janestreet.com, Jane Street, UK
‡sdolan@janestreet.com, Jane Street, UK

is explicitly targeted by Multicore OCaml. We review how these challenges are approached in OCaml with the Coq proof assistant, in Isabelle/ML, and in Haskell.

Then, we propose to reframe the issue using concepts from exception safety in C++ (Abrahams 2000) and fault tolerance (Armstrong 2003; Siedersleben 2006). We specifically borrow from Rust a concepts extending Erlang's "let it fail" approach to concurrent and parallel programming with shared mutable state (Crichton and Rust team 2015), which we recast in this context. We build on this in order to outline a design for interrupts in ML.

Lastly, devil lurks in the details as we will discuss more specific considerations for the OCaml language. Four key points of our design for OCaml are: 1) a new kind of exceptions, not caught by current catch-all handlers, 2) a separation between raising and non-raising asynchronous callbacks, 3) a straightforward design for the interruption of system calls, 4) consideration for backwards-compatibility.

We conclude with some perspective for exceptions in OCaml, and research questions for the community.

References

Abrahams, David (2000). "Exception-Safety in Generic Components". In: *Generic Programming*. Ed. by Mehdi Jazayeri, Rüdiger G. K. Loos and David R. Musser. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 69–79. ISBN: 978-3-540-39953-7.

Armstrong, Joe (2003). "Making reliable distributed systems in the presence of software errors". PhD thesis. Royal Institute of Technology, Sweden.

Crichton, Alex and the Rust team (2015). *RFC 1236: Stabilize catch_panic*. Tech. rep.

Duffy, Joe (2016). "The Error Model". In a series of blog posts on the Midori project. URL: http://joeduffyblog.com/2016/02/07/the-error-model/.

Goodenough, John B (1975). "Structured exception handling". In: *Proceedings of the 2nd ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, pp. 204–224.

Marlow, Simon et al. (Dec. 2006). "Asynchronous exceptions in Haskell". In: *ACM Conference on Programming Languages Design and Implementation (PLDI'01)*. ACM Press, pp. 274–285. URL: https://www.microsoft.com/en-us/research/publication/asynchronous-exceptions-haskell-3/.

Parnas, D. L. and H. Würges (1976). "Response to undesired events in software systems". In: *Proceedings of the 2nd International Conference on Software Engineering*. ICSE '76. San Francisco, California, USA: IEEE Computer Society Press, 437–446.

Reppy, J. H. (1990). *Asynchronous signals in Standard ML*. Tech. rep. Cornell University.

Siedersleben, Johannes (2006). "Errors and Exceptions – Rights and Obligations". In: *Advanced Topics in Exception Handling Techniques*. Ed. by Christophe Dony et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 275–287. ISBN: 978-3-540-37445-9. DOI: 10.1007/11818502_15. URL: https://doi.org/10.1007/11818502_15.

Snoyman, Michael (2018). "Asynchronous Exception Handling in Haskell". Blog post. URL: https://www.fpcomplete.com/blog/2018/04/async-exception-handling-haskell/.