# Reinventing Haskell Backtracking

Sebastian Fischer

Institut für Informatik
Christian-Albrechts Universität zu Kiel

Backtracking is an approach to explore different alternatives during search. It is often implemented in imperative programming languages by storing an explicit stack of choice points. In functional programs backtracking can be expressed by using lazy lists of results and list concatenation to express alternatives. A well-known more efficient alternative to lazy lists uses a sophisticated combination of so-called success- and failure continuations [1]. Our work sheds new light on this implementation and recasts it in order to make it more intuitively accessible.

Instead of directly coming up with success- and failure continuations, we develop them in two steps. First, we employ the concept of difference lists to overcome the inefficiency of list concatenation and then we use continuations to efficiently combine search operations sequentially. Both techniques are established functional programming folklore but they have never been applied in combination to reformulate continuation based backtracking. In fact, we were pleasantly surprised to see that he combination of difference lists with continuations leads to a well-known implementation of backtracking rather than a new one.

Our exploration of this insight lead to new implementations of other search strategies that are simpler than previous implementations in Haskell. By using specific different types instead of difference lists, our approach leads to new implementations of breadth-first- and iterative-deepening depth-first search. We obtain breadth-first search by representing levels of the search space in distinct lists, and depth-bound search by representing the search space by a function that takes a depth limit. The additional plumbing with continuations allows to separate these simple ideas from the problem of combining different searches sequentially which significantly simplifies both implementations.

The details of our approach as well as experiments to evaluate it are described elsewhere [2] and not reproduced here, due to copyright restrictions.

## References

1. Hinze, R.: Deriving backtracking monad transformers. In: ICFP '00: Proceedings of the fifth ACM SIGPLAN international conference on Functional programming, New York, NY, USA, ACM (2000) 186–197
2. Fischer, S.: Reinventing Haskell Backtracking. In: INFORMATIK 2009, Im Focus das Leben, 39. Jahrestagung der Gesellschaft für Informatik. LNI, GI (2009)