Check out this one weird trick to make your type-checker look better

Pierre-Évariste Dagand Jonathan Protzenko
INRIA
http://gallium.inria.fr/blog/

Abstract

The Mezzo programming language features an unusual type-checking discipline, in which several permissions may be available for the same variable. This means the type-checker must consider several possible types for one single variable. This precludes unification-based type-checking, and instead led us into implementing a backtracking procedure that performs forward type-checking.

The present paper extracts the essence of our derivations library. Thanks to a combination of monads, lazy streams and syntactic tricks, not only does the core of our type-checker look very close to the formal presentation of type-checking, but it also automatically builds a derivation, hence allowing the implementors to examine a successful or failed derivation.

Categories and Subject Descriptors D.2.5 [*Software Engineering*]: Testing and Debugging

Keywords functional programming, testing, quickcheck

1. Introduction

We consider the simplest possible language.

$t ::=$		type
	$t \to t$	arrow
	$()$	unit
$e ::=$		expression
	$e\ e$	application
	$\lambda x.e$	abstraction
	x	variable
	$()$	unit

Figure 1. DumbML, the degenerated language we're considering

This leads to the following OCaml definitions:

```
type typ =
  | TArrow of typ * typ
  | TUnit
```

```
type expr =
  | ELambda of string * expr
  | EApp of expr * expr
  | EUnit
  | EVar of string
```

Not only do we take an excessively simple language, but we also take an excessively stupid algorithm which, instead of using a notion of polymorphism, rather, tries several solutions. That is, when type-checking x, instead of using a unification variable and generalizing (ML) or considering one possible type (simply-typed lambda-calculus), we are going to try several possible types for x, one after another, and see which of these types "work".

The reason why are taking such a surprising approach is that, as we mentioned earlier, in Mezzo, we cannot take a classic, unification-based approach. Mezzo is at the frontier between a program logic and a type system; therefore, the usual approaches that we use when writing a type-checker no longer work. Namely, we had to compromise on the following:

- the type-checker needs to backtrack; therefore, we deal with a stream (lazy list) of solution for each expression rather than one possible type, or zero type if the expression failed to be type-checked;
- error reporting also becomes harder; right now, the derivation library we built constructs derivations or presents the caller with an explanation for the failure.

References

ICFP '14, September 1–3, 2014, Copenhagen, Denmark.

ACM 978-1-nnnn-nnnn-n/yy/mm. . . $15.00.
http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn