CS225 Spring 2018—Final Project Proposal

Kristin Mills
`git:@kmills1`
`project repository: https://github.com/kmills1/cs225-demo`

April 21, 2018

Project: Exceptions

I propose to implement a small-step semantics and type checker for simply typed lambda calculus extended with exceptions and exception handling.

Base Language I will work with the simply typed lambda calculus with booleans, natural numbers, and products as the base language.

Extended Language I will extend this language with the features of errors and exceptions. This consists of two new terms:

1. A term which raises exceptions, written `raise t`
2. An exception handler term, written `try t with t`

Applications In practical programming applications, it is common to encounter situations where a function must return information to its caller indicating that an error, or exceptional condition, has occurred which prevents it from performing its task. Some examples of these types of exceptional conditions include an array index being out of bounds, an attempted division by zero, a specified file could not be found, the system running out of memory, and a great many more situations which might cause an error case.

Sometimes it is reasonable to handle these exceptional conditions by making the function return a variant and have any callers of the function handle the possibility that these variants may be returned. However, frequently, this becomes an unruly and unrealistic task, and it becomes preferable to have an exceptional condition transfer control to an exception handler which is defined at a higher level in the program or even abort the program if there is nothing that the caller can do to handle the condition.

In addition to just indicating that an error has occurred in these instances, it is typically useful to send back some information regarding which error has occurred so that the handler can either recover and try again or provide the user with a helpful error message. In this project, the term `raise t` will be used to raise an exception, where t is the information about the type of error which will be passed back to the handler. t_2 in the exception handler term `try` t_1 `with` t_2 is a function which takes the error information as an argument. This allows this important information to be used.

Exception handling in Java is slightly different than in ML programming languages. Java uses classes rather than variants to support user-defined exceptions. Java has a built in class called `Throwable`, and any instance of `Throwable` or any of its subclasses can be used to raise an exception (`throw` in Java, which is analogous to `raise` in the case of this project) or handle an exception (`try...catch` in Java, which is analogous to `try...with` in the case of this project). Java also differentiates between the following two built-in subclasses of `Throwable`: instances of `Exception`, which the program may want to try to catch and recover from, and instances of `Error`, which indicate a seriously exceptional condition which likely cannot be handled and must result in aborting the program. In addition, defining new subclasses of `Throwable` allows a user to declare a new user-defined exception.

Project Goals For this project, I plan to complete:

1. A small-step semantics for simply typed lambda calculus extended with exceptions and exception handling
2. A type checker for simply typed lambda calculus extended with exceptions and exception handling

Expected Challenges I expect the implementation of exceptions carrying values to be challenging, as the second term in the `try...with` term must be a function taking information about the error type as an argument.

Timeline and Milestones By the checkpoint I hope to have completed:

1. A prototype implementation of the small-step semantics
2. A suite of test-cases for the small-step semantics and well-typed relation
3. One medium-sized program encoded in the language which demonstrates a real-world application of the language

By the final project draft I hope to have completed:

1. The full implementation of small-step semantics and type checking
2. A fully comprehensive test suite, with all tests passing
3. The medium-sized program running through both the semantics and type checker implementation
4. A draft writeup that explains the on-paper formalism of my implementation

5. A draft of a presentation with 5 slides as the starting point for my in-class presentation

By the final project submissions I hope to have completed:

1. The final writeup and presentation
2. Any remaining implementation work that was missing in the final project draft