CS225 Spring 2018—Final Project Proposal

Xiaosong Chen `git:xchen20`

Wyatt Wu `git:XXXwyattXXX`

April 20, 2018

Project: Exception Check of Types and Programming Languages

We propose to implement a semantics and type checker for *exception handler* which handling carefully when exception occurs very of-tern during the execution execution of a program, that disrupts the normal flow of the program's instructions.

Base Language We will work with simply typed lambda. The OCaml implementation of the first extension is *fullerror*. The language with exceptions carrying values will implement if there is enough time.

Extended Language We will extended this language with various primitives for exceptions and exception handling. This consists of new type:

1. A type variable, which carrying value for each exception, written T_{exn}

and have some four new terms:

1. Type Run-time error, written *error*
2. Type Trap Error, written *try t with t*
3. Type Raise t, written *raise t*
4. Type Handle Exceptions, written *try t with t*

Applications In situations where the exceptional conditions are truly exceptional, we may not want to force every caller of our function to deal with the possibility that they may occur. Instead, we may prefer that an exceptional condition causes a direct transfer of control to an exception handler defined at some higher-level in the program or indeed (if the exceptional condition is rare enough or if there is nothing that the caller can do anyway to recover from it) simply aborts the program.

Raising Exceptions:
When we begin design the rules for error is how to formalize abnormal termination in our operational semantics. We adopt the simple expedient of letting error itself be the result of a program that aborts. The rules in this subsection capture this behavior. This in reality means once we have error occur, no matter where is it, as long as we compile there, the is an error raise. *e.g.* in Java, if one of your methods is invoked with an invalid argument, you could throw `IllegalArgumentException`, a subclass of `RuntimeException` in `java.lang`.

Handling Exceptions:
In real implementations of languages with exceptions, this is exactly what happens: the call stack consists of a set of **activation records**, one for each active function call; raising an exception causes activation records to be popped off the call stack until it becomes empty. In most languages with exceptions, it is also possible to install exception handlers in the call stack. When an exception is raised, activation records are popped off the call stack until an exception handler is encountered, and evaluation then proceeds with this handler. In other words, the exception functions as a non-local transfer of control, whose target is the most recently installed exception handler (*e.g.*, the nearest one on the call stack).

Exceptions Carrying Values:
Talk about how the basic exception handling constructs can be enriched so that each exception carries a value. The type of this value is written T_{exn}. Roughly speaking, an exception object in Java is represented at run time by a tag indicating its class (which corresponds directly to the extensible variant tag in ML) plus a record of instance variables (corresponding to the extra information labeled by this tag).

Project Goals For this project, we plan to complete:

1. A small-step semantics for Exception
2. A type checker for Exception

Expected Challenges We wish to add up something from other chapters if time is enough. (not decide yet, but will update as soon as we figure it out)

Timeline and Milestones By the checkpoint we 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 we 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 our implementation
5. A draft of a presentation with 4-5 slides as the starting point for our in-class presentation

By the final project submissions we hope to have completed:

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