Proposal: First Class Errors

Jaraad Kamal

Introduction

For the final project I would like to implement first class errors. This includes errors that a programmer can throw and catch. I would also like to implement custom error messages that are printable and retrievable.

Language

I intend to add this feature to Loot. I will name the final language JCaml (because my name is Jaraad Kamal and it sounds like OCaml and I like puns).

Error as a Value

AST

I will create a new expression that is an Error which has a sub expression of type string. The string will be the error message. Essentially this will be a glorified string pointer.

Representation

In the compiler I would like errors to basically be a special pointer type. One the error tag is removed they will point to an address on the heap that corresponds to the string error message.

Implementation

Raising Error

This will be done through a primitive called raise. This primitive will take one expression which must evaluate to a string. The entire primitive will evaluate to an error.

Changes to Existing Compiler

Each time I compile a primitive I must check if the value in RAX is error. If it is, I need to call the ret instruction and return to the caller. This will be how I propagate the uncaught error.

The Begin expression will also need to be changed. I must check if the value from the first expression is an error and return if it is an error without executing the compiled code for expression 2.

Changes to the Interpreter

For the most part the changes to the interpreter would be similar to the changes made in the compiler. The added benefit is that the interpreter already has the error symbol that all the calls are matching against. The main change would be handling the lookup errors and the having catch not propagate up the error.

Changes to the Runtime System

I expect the only major changes made to the runtime system will be in types.h (because I am adding a new type) and in the print section so that I can properly print uncaught errors.

Implementing Try Catch

A try catch expression is just syntactic sugar of applying the following lambda expression:

```
(lambda (e1_val) (if (eq? e1_val type-error) e1_val e2))
```

Where e1 is the expression representing the try expression (e1_val is the value) and e2 is the catch expression. This means that I can convert all try catch expressions into lambda expressions under the good and reuse the lambda compile code.

Handling lookup errors.

At the moment lookup errors such as when applying functions or referencing variables happen at compile time. This would not allow us to try and catch an error. Instead of doing the racket match exception when I do not find a match, I will just add a call to raise error if the match is not found. This will then push lookup errors from compile time to runtime.

This also means that I can catch lookup errors.

Other functions:

Get message:

I will also add a function called get message. In the compiler this will return a string address that corresponds to the address that the error was pointing to. Because the error type is just an address to the string with an error tag this will just be changing the tag on the pointer.