

Comp 311

Functional Programming

Eric Allen, PhD
Vice President, Engineering
Two Sigma Investments, LLC

Try/Catch Example

Recall: Reducing a Try/ Catch

- Set aside the continuation C of the `try/catch`
- Reduce the body of the `try` in a special continuation D
- If D reduces to `throw v`:
 - Restore the continuation C
 - Try matching v against each pattern in the `catch` clause
 - If a match is found, evaluate the body of the matching case
 - Otherwise, reduce to `throw v`
- If D reduces to w , restore continuation C and reduce the `try/catch` to w

Consider Our Motivating Test Helper Function

```
def assertConstructorFail(m:Int, n:Int) = {  
  try {  
    Rational(m,n)  
    fail()  
  }  
  catch {  
    case e: IllegalArgumentException => {  
    }  
  }  
}
```

We Call Our Function In An Enclosing Context

```
enclosingProgram (  
    assertConstructorFail(1,0)  
)
```

↳

```
enclosingProgram (  
    try {  
        {require(0 != 0); Rational(1,0)}  
        fail()  
    }  
    catch {  
        case e: IllegalArgumentException => {}  
    }  
)
```

↳

Continuation C

```
enclosingProgram (  
  try {  
    {require(0 != 0); Rational(1,0)}  
    fail()  
  }  
  catch {  
    case e: IllegalArgumentException => {}  
  }  
)
```

Redex

→

```
{  
  {require(0 != 0); Rational(1,0)}  
  fail()  
}
```



→

```
{  
  {require(0 != 0); Rational(1,0)}  
  fail()  
}
```



↳

```
{  
  {throw IllegalArgumentException; Rational(1,0)}  
  fail()  
}
```

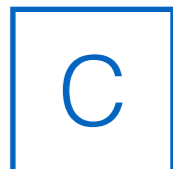


↳

```
throw IllegalArgumentException
```



↳



throw IllegalArgumentException

↳

```
enclosingProgram (
```

```
  try {
```

```
    throw IllegalArgumentException
```

```
  }
```

```
  catch {
```

```
    case e: IllegalArgumentException => {}
```

```
  }
```

```
)
```

↳

```
enclosingProgram (
```

```
  {}
```

```
)
```

↳

```
enclosingProgram ()
```


What If Our Catch Clause Does Not Match?

```
throw IllegalArgumentException
```

↳

```
enclosingProgram (  
  try {  
    throw IllegalArgumentException  
  }  
  catch {  
    case e: AssertionError => {}  
  }  
)
```

↳

```
enclosingProgram (  
  throw IllegalArgumentException  
)
```

↳

```
throw IllegalArgumentException
```

C

Continuations Are A Recurrent Concept in Computer Science

- Distributed computing
- Parallel computing
- Operating systems
- A unified approach to control flow

Some Additional Helpful Language Features

The Assert Function

```
assert: Boolean → Unit
```

```
assert: (Boolean, String) → Unit
```

- Note that the function is overloaded
- Use inside functions to ensure properties hold
- Do not assert unless you actually believe the assertion is true!

Type Checking Overloaded Functions

- For each overloaded declaration of a function f:
 - Provide that declaration with a fresh name, in a manner that respects method overriding

```
abstract class Shape {  
  def area(): Double  
  
  def makeLikeMe(that: Int): Shape  
  def makeLikeMe(that: Shape): Shape  
}
```

Type Checking Overloaded Functions

- For each overloaded declaration of a function f:
 - Provide that declaration with a fresh name, in a manner that respects method overriding

```
abstract class Shape {  
  def area(): Double  
  
  def makeLikeMe$Int(that: Int): Shape  
  def makeLikeMe$Shape(that: Shape): Shape  
}
```

Type Checking Overloaded Functions

- For each overloaded declaration of a function f:
 - Provide that declaration with a fresh name, in a manner that respects method overriding

```
case class Circle(radius: Int) {  
  val pi = 3.14  
  def area(): Double = pi * r * r  
  
  def makeLikeMe$Int(that: Int): Shape = this  
  def makeLikeMe$Shape(that: Shape): Shape = that  
}
```

Type Checking an Overloaded Function

- When an overloaded function is called on an argument expression e with type T :
 - If there is a unique matching function definition whose parameter type is:
 - A supertype of T
 - A subtype of all other matching definitions
 - Replace the function name with the unambiguous name for that unique function

Reducing an Overloaded Function Definition

- Because of the rewrite during type checking, our reduction rules need no modification!