Comp 311
Functional Programming

Nick Vrvilo, Two Sigma Investments
Robert “Corky” Cartwright, Rice University

November 11, 2018

Semantics of Exceptions

Continuations

- Reification of *what happens next*
- Captures the remainder of the computation at a given point in a computation
- Example:

$$\underbrace{\texttt{f(x, y)}}_{\text{Currently evaluating}}\ \underbrace{\texttt{+ z}}_{\text{Continuation}}$$

More Continuation Examples

- **Tail calls**
  A function call is a tail call iff the continuation of the call in the current method is empty; i.e., the continuation is returning to the parent caller.

- `if (x) y else z`
  Continuation of *x* is *y* when *x* is true, and *z* otherwise

- `f(x match {case A => {…} case B => {…}})`
  Continuation of *case A => {…}* is to call the function *f* with the resulting value

Semantics of Exceptions

- Thrown exceptions cause a sudden change in a program's flow of control
- Exceptions cause the current *continuation* to be replaced with an error handler
- The `catch` block of the closest enclosing `try` block is the current error handler (if it has a matching `case`)
- If there is no error handler, then evaluation ends in an error state with the thrown exception value

Try/Catch Blocks

```
try {
  expression₀
}
catch {
  case ExceptionPattern₁ => expression₁
  case ExceptionPattern₂ => expression₂
  …
}
```

Exception Reduction Rules

To reduce an expression `throw x`, where *x* has already been reduced to some exception value:

- Replace the entire body of the closest-enclosing `try` block with `throw x`
- If one of the `case` clauses in the corresponding `catch` block matches the exception *x*, then reduce the try/catch block to the case's expression (just like you would do for a `match` block)
- If none of the cases match, then propagate `throw x` to the next-closest enclosing `try` block
- If there are no more enclosing `try` blocks, then replace the entire remainder of the program with `throw x` as the final result

Reducing to an Error

```
require(false) ↦
throw new IllegalArgumentException()


1 / 0 ↦
throw new ArithemeticException()


{
  val x: List[Int] = Nil
  val List(y, z) = x
  …
} ↦
throw new MatchError()
```

Try/Catch Example

```
100 +
try {
  try {
    5 + 1 / 0
  }
  catch {
    case _: AssertionError => -1
    case _: MatchError => -2
  }
}
catch {
  case _: Exception => -3
}
```

Try/Catch Example

```
100 +
try {
  try {
    5 + throw new ArithmeticException()
  }
  catch {
    case _: AssertionError => -1
    case _: MatchError => -2
  }
}
catch {
  case _: Exception => -3
}
```

Try/Catch Example

```
100 +
try {
  try {
    throw new ArithmeticException()
  }
  catch {
    case _: AssertionError => -1
    case _: MatchError => -2
  }
}
catch {
  case _: Exception => -3
}
```

No matching case clause

Try/Catch Example

```
100 +
try {
  throw new ArithmeticException()
}
catch {
  case _: Exception => -3
}
```

Matching case clause

Try/Catch Example

```
100 + { -3 }   ↦ 97
```

Expressions that *Throw*

- ArithmeticException: divide by zero
- NoSuchElementException:
  `Nil.head, Map(1→2).apply(3), …`
- ArrayIndexOutOfBoundsException
- MatchError
- AssertionError: `assert`, `ensuring` clause failures
- IllegalArgumentException: `require` clause failure

Implicit Conversions

https://docs.scala-lang.org/tour/implicit-conversions.html

Value Classes

https://docs.scala-lang.org/overviews/core/value-classes.html