# Scala Continuations meet JVM Coroutines

Miles Sabin, Chuusai Ltd.
http://www.chuusai.com/

http://uk.linkedin.com/in/milessabin
http://twitter.com/milessabin

# Outline

- Background
- Continuations
  - Varieties of continuation
  - Shift/reset in Scala
- Coroutines and generators in Scala
- Limitations of Scala continuations
- JVM coroutines

# Background

# Continuations

# Varieties of continuation

- A continuation captures "the remainder of the computation"
- Two main axes of variation
  - Full (whole programme) vs. delimited
  - Direct implementation via stack manipulation vs. CPS transformation
- Full continuation support is quite common
  - call/cc or equivalent in Lisps, Smalltalk, ML, setjmp/longjmp in C/C++

# Varieties of continuation

- Continuations have traditionally been used as primitives for defining higher-level control constructs
  - Concurrency constructs
  - Exception handling
- They offer a lot more, particularly in a typed setting (but we won't have time for that today)

# Shift/reset in Scala

- Scala offers a form of delimited continuation implemented in terms of a localized CPS translation
  - "delimited" here means that we are capturing "the remainder of the computation" up to a particular marker
  - The CPS transformation implementation allows it to run efficiently on a standard JVM
  - Localized transformation supports interop with ordinary code (up to a point ...)

# Shift/reset in Scala

- Two complementary constructs,

  `reset { ... }` Delimits the continuation

  `shift { (cont : ...) => ... }`
  Captures the continuation up to the enclosing reset

- shifts must be (dynamically) enclosed within a corresponding reset
  - This is enforced by an effect type system

# Shift/reset in Scala

- Continuations support is provided via a compiler plug-in
- This ships with the complier as standard but isn't enabled by default
- It can be used with both the compiler and the REPL

```
scalac -P:continuations:enable ...
scala -P:continuations:enable
```

# Coroutines and generators in Scala

# Coroutines

- Coroutines are an imperative control structure
- They can be viewed as subroutines with multiple entry and exit points
- They interleave computations, and can replace threads in some applications
  - Very lightweight relative to threads
  - No actually concurrency ... interleaving is on a single thread

# Generators

- Related to coroutines
- Main application is the production of sequences of values, lazily, on demand
- This can be done using traditional iterators or streams, but generators are able to use the stack to manage state implicitly
  - This can often significantly simplify the implementation

# Limitations of Scala continuations

# Limitations

- CPS effect types both a blessing and a curse
  - All paths from shift to dynamically enclosing reset must be CPS aware
  - This means we can't shift from within any non-CPS aware Scala framework (could effect type polymorphism help here?)
  - This means we can't shift from within any Java frameworks at all

# Limitations

- Effect types don't play nicely with CPS effect types
  - for comprehensions – workarounds
  - Pattern matching – Scala virtualized
- The behaviour of transformed code can be hard to reason about given only untransformed source (@tailrec)

# JVM coroutines

# JVM coroutines

- Development of work by Lukas Stadler at the Johannes Kepler University, Linz
- Implementation is part of the Da Vinci Machine Project – patch available from MLVM repository
- JSR recently formed to promote it (I'm a member) for Java 8

# JVM coroutines

- The aim is to provide the minimum infrastructure to support interesting control flows
- No bytecode changes – library additions only (but Hotspot level support)
- Implementation is via stack manipulation (a hybrid of stack copying and spaghetti stacks)

# JVM coroutines

- JVM coroutines come in two flavours
  - Instances of class Coroutine,
    - These are executed by per-thread schedulers
    - Control transfer is via an explicit yield
    - Somewhat thread-like semantics
  - Instances of class AsymCoroutine,
    - These may be invoked repeatedly with an argument returning a result
    - May only return to their caller
    - Generator-like semantics (implement j.u.Iterator)

# JVM coroutines

- The Java API is a bit clunky, but it's relatively straightforward to provide a Scala shim
- All of the previous limitations are lifted
  - We can yield from within any framework
  - for comprehensions and pattern matching are fully supported
  - Control flow is more transparent

# JVM coroutines

- What's not to like?
  - Context switch time is very good

| | |
|---|---|
| Scala Stream | 1200ns |
| Scala Generator | 100ns |
| Java Thread | 16000ns |
| JVM Coroutine | 80ns |

  - Memory footprint is not so good

| | |
|---|---|
| Scala Stream | 1.5k |
| Scala Generator | 0.5k |
| Java Thread | 64k |
| JVM Coroutine | 48k |

# Scala Continuations meet JVM Coroutines

Miles Sabin, Chuusai Ltd.
http://www.chuusai.com/

http://uk.linkedin.com/in/milessabin
http://twitter.com/milessabin