Composition of Coroutines

Stephen Diehl

October 19, 2012

CRS

- CRS Semantics
  - Create
  - Resume
  - Suspend
- Tasks that can be suspended and resumed.
- State can be passed between context switches.

Generators

```
def co1(i):
    yield 1
    yield 2

def co2(i):
    yield "foo"
    x = i.next()
    y = i.next()
    yield x+y

it = co2(co1(None))
for i in it:
    print i
```

```
foo
3
```

Streams

- enumerators / iterators
- generators
- iteratees
- pipes / conduits
- coroutines

Composition

- Synchronization
- Resource management
- Pure vs Impure

Imperative Implementations

- multi-shot vs. single-shot
- symmetric vs. asymmetric
- stackful vs. non-stackful

Python Implementations

- Generators
  - Asymmetric
  - Single-shot
  - Non-stackful
- Enhanced Generators
  - Symmetric
  - Single-shot
  - Non-stackful
- Greenlets
  - Symmetric
  - Single-shot
  - Stackful
- Tasklets
  - Symmetric
  - Multi-shot
  - Stackful

Greenlets

```
from greenlet import greenlet, getcurrent

def coro(x):
    getcurrent().parent.switch(x+1)

gr = greenlet(coro)
gr.switch(1)
```

Assembly Fun

```
void *ebp, *ebx;
unsigned short cw;
register int *stackref, stsizediff;
__asm__ volatile ("" : : : "esi", "edi");
__asm__ volatile ("fstcw %0" : "=m" (cw));
__asm__ volatile ("movl %%ebp, %0" : "=m" (ebp));
__asm__ volatile ("movl %%ebx, %0" : "=m" (ebx));
__asm__ ("movl %%esp, %0" : "=g" (stackref));
{
    SLP_SAVE_STATE(stackref, stsizediff);
    __asm__ volatile (
        "addl %0, %%esp\n"
        "addl %0, %%ebp\n"
        :
        : "r" (stsizediff)
        );
    SLP_RESTORE_STATE();
```

Haskell Inspiration

```haskell
data Id x = Id x
newtype (Compose g f) x = C { unCompose :: g (f x) }

instance Functor Id where
    fmap f (Id x) = Id (f x)

instance (Functor f, Functor g) => Functor (Compose g f) wh
    fmap f (C xs) = C $ fmap (fmap f) xs
```

Haskell Inspiration

```haskell
import Control.Category

class Category cat where
    id :: cat a a
    (.) :: cat b c -> cat a b -> cat a c

newtype Coroutine i o = Coroutine {
    runC :: i -> (o, Coroutine i o)
}

instance Category Coroutine where
    id = Coroutine $ \i -> (i, id)

    a . b = Coroutine $ \i ->
        let (x, a') = runC b i
            (y, a') = runC a x
        in (y, a' . b')
```

Pipes

Composition

$$a \gg b$$

Identities

$$\text{idP}$$

Associative

$$(a \gg b) \gg c = a \gg (b \gg c) = a \gg b \gg c$$

$$a \gg \text{idP} = \text{idP} \gg a = a$$

Flowlets

Greenlets with extra "structure".

```
@flowlet
def A():
    send(1)
    send(2)

@flowlet
def B(x):
    send(x)
    y = await()
    z = await()
    send(x+y+z)

>>> line = A() >> B(1)
>>> runPipeline(line)
[1, 4]
```

Flowlets

```
@flowlet
def A():
    for i in count(0):
        send(i)

@flowlet
def B():
    for i in xrange(3):
        t = (await(), await())
        send(t)

>>> A() >> B()
[(0,1), (2,3), (3,4)]
```

Flowlets

```
from itertools import count

line = count(1) >> pipe(lambda x:x**2) >> take(3)

>>> runPipeline(line)
[1, 4, 9]
```

Resources

```
import socket
from flowlet.prelude import closing

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 8000))

@flowlet
def A(sock):
    with closing(sock):
        msg = sock.recv(1024)
        send(msg)

line = A(s)
>>> runPipeline(line)
```

Parallel Pipes

```
f = a >> b >> c
g = x >> y >> z

line = I >> par(f, g) >> O

>>> runPipeline(line)
```

Ideas

Composition semantics are well defined.

Abstracts away the complexity of creating a parallel pipeline so that you can focus on the internal operations of the pipeline instead of the composition edge cases.

Semantics of parallel composition is the same as single threaded composition.

Questions

- Questions / Comments ?
- Also, I do consulting.

Internal Logic

```
(a >> b)
  -- A = \x,y -> y(b(a(x)))
(a >> b) >> c
  -- \x,y -> A(x, c) = \x,y -> y(c(b(a(x))))
((a >> b) >> c) >> d
  -- \x,y -> A(x, d) = \x,y -> y(d(c(b(a(x)))))
```

```
(c >> d)
  -- A = \x,y -> y(d(c(x)))
( b >> ( c >> d ) )
  -- \x,y -> A(b, x) = \x,y -> y(c(b(a(x))))
(a >> ( b >> ( c >> d )))
  -- \x,y -> A(a, x) = \x,y -> y(d(c(b(a(x)))))
```