ssio

Simple Semantics Input Output

from the boostcon '10 FP team

netsuperbrain.com/ssio

zipped src and mercurial repo

Strategy

- Design semantics using math and critique.
- Implement the Math Efficiently

The Math

Sk a = a → Action

Src a = Sk a → Action

put : Sk a → a → Action

put(sk, a) = sk(a)

step : Src a → Sk a → Action

step(src, sk) = src(sk)

Flt a b = Sk a → Sk b

apSk : Flt a b → Sk a → Sk b

apSk (f, sk) = f(sk)

apSrc : Flt b a → Src a → Src b

apSrc (f, src, sk)

= step(src, f(sk))

UnParser a b = b → Src a

Parser a b = Flt b a

Implementation

- Don't implement math directly.
- Use concepts for our main types, Sources, Sinks, and Filters.

Efficient Implementation (thanks Robert for ASM output!)

```
put( console, 'a' );
    step( keyboard, console );
```

```
movq    stdout(%rip),
  %rsi
  movl    $97, %edi
  call    _IO_putc
  movq    stdin(%rip),
  %rdi
  call    _IO_getc
  movq    stdout(%rip),
  %rsi
  movsbl  %al,%edi
  call    _IO_putc
```

Expressive Algebra

```
// console is a sink of type char
   // lines is a filter from char to string
   // lineConsole is a sink of type string
   auto lineConsole = apSink( lines, console );

put( lineConsole, "This is a line" );
// Console displays “This is a line\n”
```

Simple to make efficient components

```
struct HelloWorld { };
const HelloWorld helloWorld = HelloWorld();

template<>
struct source<HelloWorld>
{
    typedef std::string type;

    template<typename Sink>
    static void step( const HelloWorld, const Sink & s )
    {
        put( s, std::string("hello world") );
    }
};
```

Math allows us to be fancy

- Every filter can work on sources or sinks

  step(apSource(lines, helloWorld), console);

- Filters can be composable (not implemented yet) [Mikhail's idea]

  compF(filter1, filter2)

Implementation allows us to be wicked Fast

- The puts function could be a concept with a default naive implementation, but can be optimized for certain sinks.
- Other, unforeseeable optimizations are possible without breaking the original semantics [buffering, etc.].

Future Directions

- Implement get for sources
- Add example buffering/mutex filters.
- Work on semantics for seekable sources: Seekable a = (Int, Int → a)?
- Think about exhaustible sources;
  Exaustable a = Src (a or nothing)?
- Make a less efficient, but easy to extend, template compression version of structures.