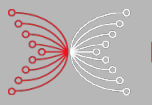

BENCHMARKING

Micro-benchmarks on monad evaluation

The bracket around an IO action will enter a named context, evaluate the action, and return its result.

```
bracketObserveIO :: Configuration
     -> Trace IO a -> Severity -> Text
     -> IO t
     -> IO t
```

The set of observed counters are traced as `ObserveOpen`, `ObserveClose` and `ObserveDiff` with the indicated severity to the *Trace*.
An exception thrown in the action will be traced and rethrown.

Other bracketing functions exist:

```
bracketObserveM :: (MonadCatch m, MonadIO m)
  => Configuration -> Trace m a -> Severity
  -> Text -> m t -> m t
bracketObserveX :: (MonadIO m)
  => Configuration -> Trace m a -> Severity
  -> Text -> m t -> m t
```

Benchmarking STM transaction

A bracket, in a named context, can be placed around a STM action:

```
bracketObserveIO :: Configuration
     -> Trace IO a -> Severity -> Text
     -> STM t
     -> IO t
```

This will return the result from successfully evaluating the STM action, which does not have access to logging.

A second bracket function also traces the log items (pairs of meta data and content) which are output by the STM action and returns its result.

```
bracketObserveLogIO :: Configuration
    -> Trace IO a
    -> Severity
    -> Text
    -> STM (t,[(LOMeta, LOContent a)])
    -> IO t
```

Aggregation

Observables can be forwarded for aggregation, which currently can aggregate them into a basic statistics or an exponentially weighted moving average (EWMA).
Aggregated values reenter the switchboard with '#aggregation' prepended to their name, and can be routed like ordinary messages.

Values may be of type:

```
Measurable = Microseconds  | Nanoseconds | Seconds
           | Bytes         | Severity
           | PureI Integer | PureD Double
```

The statistics computes: minimum, maximum, mean, std. dev., and count of (1) the observed values, (2) their differences to the previous, and (3) the time between messages.

Observables - OS counters

platform independent:

`MonotonicClock`	clock with µs precision
`GhcRtsStats`	Haskell RTS values (gc, mem)

Linux specific:

`MemoryStats`	reports memory usage
`ProcessStats`	lots of process info: cpu, mem, io, ..
`IOStats`	block device I/O
`NetStats`	network I/O

The Linux specific counters for the current process are obtained from the `/proc` interface into the kernel.

To trace observables, the configuration needs to find a definition of a subtrace `ObservableTrace` for the context name. Only the mentioned counters in the list will be recorded.

```
CM.setSubTrace
    config
    "proc.observed"
    (Just $ ObservableTrace observablesSet)
  where
    observablesSet = [MonotonicClock, MemoryStats]
```

A complete example of bracketing monadic and STM actions is in example-complex.

Run it with `cabal new-run example-complex`
or `stack run example-complex`