Problem Wk.4.3.3: State Machine Composition

Use `sm.R`, `sm.Gain`, `sm.Cascade`, `sm.FeedbackAdd` and `sm.FeedbackSubtract` to construct the following state machines. These are the same systems that appeared in Wk.4.2.1.

Define a Python procedure called `accumulator` that takes one argument:

- the output before the first input, `init`.

and which returns a state machine whose output at time `n` is the sum of its inputs up to and including time `n`. So the output at time 0 is the sum of `init` and the input at time 0.

Define a Python procedure called `accumulatorDelay` that takes one argument:

- the output at time 0, `init`.

and which returns a state machine whose output at time `n` is the sum of its inputs up to and including time `n-1`.

Define a Python procedure called `accumulatorDelayScaled` that takes two arguments:

- the scale factor `s`.
- the output at time 0, `init`.

and which returns a state machine whose output at time `n` is the sum of the scaled inputs (each scaled by `s`) up to and including time `n-1`.

You can debug these in Idle. In the design lab folder create a file with

```
import lib601.sm as sm
```

at the top. You can then place your definitions and test cases in the file. Evaluate them with Run Module. You can use `transduce` to test your machines.

```
# Use sm.R, sm.Gain, sm.Cascade, sm.FeedbackAdd and sm.FeedbackSubtract
# to construct the state machines
def accumulator(init):
    pass
def accumulatorDelay(init):
    pass
def accumulatorDelayScaled(s, init):
    pass
```

MIT OpenCourseWare
http://ocw.mit.edu

6.01SC Introduction to Electrical Engineering and Computer Science

Spring 2011

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.