```
------------------------------------------------------------
Lab: Sequential Machines
Author: Roger Hui
To advance the lab, select menu Studio|Advance or the
corresponding shortcut.
```

```
-- (1 of 17) Introduction ---------------------------------

x;:y implements a sequential machine (finite state machine,
finite state automaton).   x is the specification of a
machine, including the state transition table, and y is the
input.  A sequential machine solves the problem of
recognizing the "words" in the input.  The machine starts in
some initial state and processes the input one item at a time.
Given the current state and input item, the new state and
output are determined by the state transition table.  The
machine then proceeds to the next input item.
```

```
-- (2 of 17) Introduction (ctd) ---------------------------

x=.f;s;m;ijr  is a boxed list of the specifications for a
machine.

f is a function code.  m is an input mapping.  ijr are the
initial settings.  ijr or both m and ijr may be elided.

s is a state transition and output table ("state table" for
short).  s is actually a 3-dimensional array with shape
(p,q,2) where p is the number of states and q is the number
possible mapped inputs.  However, it is convenient to speak
of s as a p by q table of pairs.  The pairs are 2 integers
denoting the new state and the output code.

The arguments are described in greater detail in later
sections of this lab.
```

```
-- (3 of 17) Example: Cut ---------------------------------

The following example mimics the functionality of the monad
<;.1 (cut).  The space specifies a cut point, and each cut
(word) is boxed.

The input mapping is ' '=a. -- the space character is mapped
to 1 and other characters are mapped to 0.

The state table has 2-columns (2 possible mapped inputs) and
2 rows.

Column 0 is for "other" (non-space) and column 1 is space.

Row 0 is the initial state.  The sequential machine persists
in this state until a space is scanned.  When that happens,
a new word is started and then it goes into and remains in
state 1.  In state 1, if a non-space is scanned, there is no
output; if a space is scanned, the current word is output
and a new word begins.  At the end, the then current word is
output.
```

```
Can you define a sequential machine such that the space is
excluded from the word?
```

```
   sc=: 2 2 2$ 0 0  1 1  1 0  1 2  NB. state table
   <"1 sc
+---+---+
|0 0|1 1|
+---+---+
|1 0|1 2|
+---+---+
   y=: ' fourscore and seven years ago, our fathers'
   (0;sc;' '=a.) ;: y
+----------+----+------+------+-----+----+--------+
| fourscore| and| seven| years| ago,| our| fathers|
+----------+----+------+------+-----+----+--------+
   (0;sc;' '=a.) ;: 'junk@front zero  two'
+-----+-+----+
| zero| | two|
+-----+-+----+
```

```
-- (4 of 17) Example: Cut (ctd) ----------------------------

A benchmark comparing the sequential machine and the cut
primitive demonstrates the efficiency of the sequential
machine approach.
```

```
   f=: (0;sc;' '=a.)&;:  NB. sequential machine method
   g=: <;.1              NB. cut primitive
   y=: 1e6$' fourscore and seven years ago, our fathers'
   (f -: g) y
1

   ts=: 6!:2 , 7!:2@]    NB. time and space
   ts 'f y'
0.110595 1.42287e7
   ts 'g y'
0.091472 1.21316e7
```

```
-- (5 of 17) The Sequential Machine Computation -------------

Certain quantities are helpful in understanding the
sequential machine computation.  These are:

 i          iteration index (also input index)
 j          beginning index of the current word
 r          current state
 c          current mapped input
 0{s{~<r,c  new state
 1{s{~<r,c  output code

By default:  i runs from 0 to (#y)-1.  The word index j is
initialized to _1 and is set as specified by output codes in
the state table, according to the current state r and the
current mapped input c.  Likewise, the current state r is
initialized to 0 and is set as specified by the state table,
according to the current state and the current mapped input
c.
```

```
Non-default values for i, j, and r can be specified in last
item of the left argument.  (See a later section of this
lab.)
```

```
-- (6 of 17) Arguments -------------------------------------

As outlined briefly, the left argument x is  f;s;m;ijr  and
the right argument y is the input.  These are described in
detail in the following sections.
```

```
-- (7 of 17) Input and Mapped Input -------------------------

The right argument y is the input.  y is usually a string but
can be any array.

Item 2 of the left argument x (m=.>2{x) is a mapping on the
input.  For example, in the "cut" example in a previous
section, the input is a string and the mapping is ' '=a. --
i.e. the space character is mapped to 1 and all other
characters are mapped to 0.  Typically, in working with
sequential machines, the first items to be decided are the
input and the input mapping.

If y is a string, then m is usually a 256-element integer
list, where each element specifies an integer value for
the corresponding character.  (e.g. ' '=a.)

In general, m is a boxed list, where >i{m contains the items
that should be mapped to the integer i .  For example,
m=.'aeiouy';'bcdfghjklmnpqrstvwxz'  specifies that the
letters a e i o u y are mapped to 0, other lower case letters
are mapped to 1, and all other characters are mapped to 2.
The mappings  ' '=a.  and  <a.-.' '  are equivalent.

If m is elided or is the empty vector, then the input y is
used as is.  In that case y must be an integer list.

The state transitions and output are determined by the
mapped input, but the output words are formed from the
original input.
```

```
-- (8 of 17) State Transition and Output Table --------------

Item 1 of the left argument x (s=.>1{x) is a state transition
and output table ("state table" for short).  s is actually a
3-dimensional array with shape (p,q,2) where p is the number
of states and q is the number possible inputs.  However, it
is convenient to speak of s as a p by q table of pairs.
The pairs are 2 integers denoting the new state and the
output code.

q, the number of columns in the state table, must bound the
input mapping values.

Typically, in working with sequential machines, the state
table is the next to be devised after the input mapping.
Start with a table of with q columns and as many rows
(states) as one can think of.  The new state and output need
```

```
to be decided for each state and mapped input combination.
In the process, quite often additional states are added.

Sometimes the state table is sufficiently complicated that
it is computed by a program from other information.  The
Huffman coding lab has an example of this.
```

```
-- (9 of 17) Output Codes ----------------------------------

The second element of the pairs in a state table is an output
codes, one of the integers from 0 to 6.

 0    no output
 1    j=.i
 2    j=.i  [ ew(i,j,r,c)
 3    j=._1 [ ew(i,j,r,c)
 4    j=.i  [ ev(i,j,r,c)
 5    j=._1 [ ev(i,j,r,c)
 6    stop

j is the beginning index of a word and is initialized to _1
(or 1{ijr if ijr is supplied).

ew(i,j,r,c) ("emit word") checks that j is not _1 and emits
information on a word according to the function code f .

ev(i,j,r,c) ("emit vector") is similiar, but the current word
is catenated to the previous word if the previous emit was ev
and the state at that time was r .  ev is used to form
"vector constants" and is not used in most applications.
```

```
-- (10 of 17) Function Codes -------------------------------

The function code f (the first item in the left argument) is
one of the integers from 0 to 5.  0 to 4 specify the
treatment of a word at the time of output.  5 specifies
trace.

 0  <y{~j+i.i-j      the word boxed
 1  y{~j+i.i-j       the word
 2  j,i-j            word index and length
 3  c+q*r            state table index
 4  j,(i-j),c+q*r    both 2 and 3 above
 5  i,j,r,c,s{~<r,c  trace

Typically, function code 0 would be used if the words
themselves are of interest.

Function code 1 would be used if some words are retained
while others are discarded (as in the quote/non-quote
example in a later section of this lab).

Function codes 2, 3, or 4 are used if the result of x;:y
is used for further computation.  For example, in Huffman
decoding it is convenient to use function code 3 (see the
Huffman Coding lab).

Function code 5 can be useful for debugging.  It provides
a trace of the sequential machine computation.  The result
```

```
is a 6-column table of quantities that completely describe
the sequential machine computation.

The examples below use the state table sc (cut on space)
from a previous section.
```

```
   (0;sc;' '=a.) ;: y=: ' tonic chthonic'
+------+---------+
| tonic| chthonic|
+------+---------+
   (1;sc;' '=a.) ;: y
 tonic chthonic
   (2;sc;' '=a.) ;: y
0 6
6 9
   (3;sc;' '=a.) ;: y
3 2
   (4;sc;' '=a.) ;: y
0 6 3
6 9 2
   (5;sc;' '=a.) ;: y
 0 _1 0 1 1 1
 1  0 1 0 1 0
 2  0 1 0 1 0
 3  0 1 0 1 0
 4  0 1 0 1 0
 5  0 1 0 1 0
 6  0 1 1 1 2
 7  6 1 0 1 0
 8  6 1 0 1 0
 9  6 1 0 1 0
10  6 1 0 1 0
11  6 1 0 1 0
12  6 1 0 1 0
13  6 1 0 1 0
14  6 1 0 1 0
```

```
-- (11 of 17) Initial Settings ------------------------------

The initial settings ijr (>3{x) are 3 integers:

 i  iteration index (also input index)
 j  beginning index of a word
 r  initial state

If ijr is elided, then the defaults are 0 _1 0 .
```

```
-- (12 of 17) Argument Checking -----------------------------

The verb smcheck below may be helpful in checking the
arguments x and y in x;:y for errors.

If x smcheck y does not signal error, then the only logic
error that can occur in x;:y is "index error" -- the word
index j is _1 when outputting a word.  ("out of memory",
"break", and "attention interrupt" can also be signalled.)
```

```
   type=: 3!:0  NB. internal type
```

```
   smcheck=: 4 : 0     NB. check arguments of sequential machine
 assert. 32=type x.
 assert. (<$x.) e. ,&.>2 3 4    NB. 2- or 3- or 4-element list
 'f s m ijr'=. 4{.x.            NB. function code; state ; mapping; initial state
 assert. (0=#$f) *. f e. i.6    NB. function code is from 0 to 5
 assert. (3=#$s) *. 2={:$s      NB. state is a 3-d, 2-column array
 assert. (0<:s) *. s-:<.s       NB. positive integers
 assert. (#s) > >./,0{"1 s      NB. new states
 assert. (1{"1 s) e. i.7        NB. output codes
 assert. 1=#$m                  NB. mapping is a vector
 q=. 1{$s
 if. ''-:m do.                  NB. if m is empty
  assert. (1=#$y.) *. y.-:<.y.  NB. y is used as is
  assert. q > y.
 elseif. (1:@:+: :: 0:) m do.   NB. if m is numeric
  assert. q > m
  assert. m -:&$ a.             NB. m specifies mapping on alphabet
  assert. m e. i.#a.            NB. mappings are non-negative integers
  assert. (1=#$y.) *. 2=3!:0 y. NB. y must be a string
 elseif. 1 do.
  assert. 32=type m             NB. else m must be boxed
  assert. q > #m
  assert. (#$;m) e. 0 1+#$y.
 end.
 if. -. ''-:ijr do.             NB. ijr may be elided or empty
  assert. (,3)-:$ijr            NB. 3-element vector
  assert. ijr-:<.ijr            NB. integers
  'i j r'=. ijr
  assert. (0<:i)*.i<i.#y.       NB. iteration index (also index into input)
  assert. (_1=j)+.(0<:j)*.j<i   NB. beginning index of a word
  assert. (0<:r)*.r<p           NB. initial state
 end.
 1

   (1;sc;''''=a.) smcheck 'abc'
1
   (9;sc;''''=a.) smcheck 'abc'  NB. deliberate error
|assertion failure: smcheck
|   (0=#$f)*.f e.i.6
```

```
-- (13 of 17) Example: English Words ------------------------

A sequential machine that selects the words in ordinary
English text is similar to the machine in a previous section
that cuts on words separated by spaces.

The mapping is 1 for letters of the alphabet (majuscules and
minuscules) and 0 for everything else.
```

```
   se=: 2 2 2 $ 0 0  1 1  0 3  1 0
   <"1 se
+---+---+
|0 0|1 1|
+---+---+
|0 3|1 0|
+---+---+
   f=: (0;se;(i.#a.) e. ,(a.i.'Aa')+/i.26)&;:
```

```
   f 'In the beginning, E=mc^2'
+--+---+---------+-+--+
|In|the|beginning|E|mc|
+--+---+---------+-+--+
```

```
-- (14 of 17) Example: Quotes and Non-quotes ----------------

The following example separates quotes from non-quotes.

The input mapping is ''''=a. -- the quote character is
mapped to 1 and other characters are mapped to 0.

The state table has 2-columns (possible mapped inputs) and 4
rows.  Column 0 is for "other" (non-quotes) and column 1 is
quotes.  The rows (states) are:
 0  initial state
 1  outside of quotes
 2  within a quote
 3  within a quote and a quote is seen

For example, suppose the current state is 1 (outside of
quotes).  If the current mapped input is 0 (other), then
the relevant state table entry is 1 0: the new state is
1 and the output code is 0 (no output).  On the other hand,
if the current mapped input is 1 (quote), then the relevant
state table entry is 2 2: the new state is 2 (within quotes)
and the output code is also 2 (emit the current word) and
start a new word.

State 3 is for handling the convention whereby two quotes
within a quote are interpreted as a single quote, and not
terminating the quoted string and immediately starting a new
quoted string.

The function code is 0 -- box the words.
```

```
   sq=: 4 2 2$ 1 1 2 1  1 0 2 2  2 0 3 0  1 2 2 0
   <"1 sq
+---+---+
|1 1|2 1|
+---+---+
|1 0|2 2|
+---+---+
|2 0|3 0|
+---+---+
|1 2|2 0|
+---+---+
   ] y=: '''The Power of the Powerless'' by Havel and ''1984'' by Orwell'
'The Power of the Powerless' by Havel and '1984' by Orwell
   (0;sq;''''=a.) ;: y
+---------------------------+-------------+------+----------+
|'The Power of the Powerless'| by Havel and |'1984'| by Orwell|
+---------------------------+-------------+------+----------+
   ] y=: '''Don''''t tread on me!'' He said.'
'Don''t tread on me!' He said.
   (0;sq;''''=a.) ;: y
+--------------------+---------+
|'Don''t tread on me!'| He said.|
+--------------------+---------+
```

```
-- (15 of 17) Example: Quotes and Non-quotes (ctd) ----------

If the desired result is the text outside of quotes, then
function code 1, the words not boxed, should be used.
As well, the output code for quotes is specified in such a
way that the quoted words are not output (not catenated).

The traditional method for this computation is based on the
not-equal scan.  The primitives used in the not-equal scan
method are individually highly optimized in the interpreter.

A benchmark comparing the two methods demonstrates the
efficiency of the sequential machine approach.
```

```
   sqx=: 4 2 2 $ 1 1 2 0  1 0 2 3  2 0 3 0  1 1 2 0
   <"1 sqx
+---+---+
|1 1|2 0|
+---+---+
|1 0|2 3|
+---+---+
|2 0|3 0|
+---+---+
|1 1|2 0|
+---+---+
   (":<"1 sq) ,. ' ' ,. (": <"1 sqx)  ,. ' ' ,. ": <"1 sq=sqx
+---+---+ +---+---+ +---+---+
|1 1|2 1| |1 1|2 0| |1 1|1 0|
+---+---+ +---+---+ +---+---+
|1 0|2 2| |1 0|2 3| |1 1|1 0|
+---+---+ +---+---+ +---+---+
|2 0|3 0| |2 0|3 0| |1 1|1 1|
+---+---+ +---+---+ +---+---+
|1 2|2 0| |1 1|2 0| |1 0|1 1|
+---+---+ +---+---+ +---+---+

   ] y=: '''The Power of the Powerless'' by Havel and ''1984'' by Orwell'
'The Power of the Powerless' by Havel and '1984' by Orwell
   (1;sqx;''''=a.) ;: y  NB. catenated words
 by Havel and  by Orwell

   f=: (1;sqx;''''=a.)&;:       NB. sequential machine method
   g=: (+: ~:/\)@(''''&=) # ]   NB. not-equal scan method
   y=: 1e6$y
   (f -: g) y
1

   ts 'f y'
0.0242 1.04941e6
   ts 'g y'
0.0182928 3.14656e6
```

```
-- (16 of 17) Example: J sentences --------------------------

The dictionary entry on ;: contains a sequential machine
implementation of word formation on J sentences, using a
```

```
10 by 9 state table.

If the rhematic rules are restricted so that NB. has no
special significance, then a 6 by 7 state table suffices.
```

```
   mjx=: ' ';(a.{~,65 97+/i.26);'0123456789_';'.';':';''''
   t=. 0 7 2$0
   NB.        S    A    9    D    C    Q    X
   t=.t,_2]\ 0 0  2 1  3 1  1 1  1 1  4 1  1 1  NB. 0 space
   t=.t,_2]\ 0 3  2 2  3 2  1 0  1 0  4 2  1 2  NB. 1 other
   t=.t,_2]\ 0 3  2 0  2 0  1 0  1 0  4 2  1 2  NB. 2 alphanumeric
   t=.t,_2]\ 0 5  3 0  3 0  3 0  1 0  4 4  1 4  NB. 3 numeric
   t=.t,_2]\ 4 0  4 0  4 0  4 0  4 0  5 0  4 0  NB. 4 quote
   t=.t,_2]\ 0 3  2 2  3 2  1 2  1 2  4 0  1 2  NB. 5 even quotes
   sjx=: t

   f=: (0;sjx;<mjx)&;:
   f y=: '(2*a) %~ (-b) (+,-) %: (*:b)-4*a*c'
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|(|2|*|a|)|%|~|(|-|b|)|(|+|,|-|)|%:|(|*:|b|)|-|4|*|a|*|c|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   (f -: ;:) y
1
   (f -: ;:) '1 2 3 +/ . * 4 5 6'
1
   (f -: ;:) 'gm=: */ %:~ #'
1
```

```
-- (17 of 17) Other Examples --------------------------------

See the lab "Huffman Coding" for another example of using
sequential machines.
```