

Restartable Scanning and Parsing

(* WORKING DRAFT *)

Brian Beckman

5 Dec 2016

This document describes stateful, restartable scanners and parsers. The design in this document separates buffer management from recognition. Buffer management includes the saving of validated characters, rewinding over invalidated characters, keeping the values of matched forms, resynchronizing the scanner when data have been missed, stuttered, or garbled, and transferring characters from input buffers to scanner and parser state variables.

This is an executable design or reference implementation for C code, but it's written in *Mathematica* for convenience of visualization and proofs. The reference implementation is intended to be easy to transcribe into C with high assurance, taking no special advantage of *Mathematica* functionality. *High assurance* means that the transcription is:

- testable, exhaustively and / or statistically
- either automated or so straightforward that the opportunities for manual error are few and easily correctable by a POSITA (Person of Ordinary Skill In The Art)

Motivating Example

We need a *scanner* (i.e., *lexical analyzer* or *recognizer*) that can be entered and exited repeatedly in the midst of recognizing a syntactically correct sequence of bytes and can restart in the midst of certain kinds of noise. Syntax may be defined by *regular expressions*, or, more generally, by a *formal grammar* that corresponds to a *deterministic finite automaton* (DFA) and a *finite state machine* (FSM). Generally, we use the term *scanning* for matching regular expressions and the term *parsing* for matching more general formal grammars. For convenience and brevity in this document, we use the terms *scanner* and *scanning* for all kinds of formal grammars.

The reason that a scanner must be restartable is that portions of syntactically correct sequences arrive at multiple times. Each *time* corresponds to a single root activation of a scanner routine.

We represent FSMs as *directed graph* structures.

A candidate syntactic unit may contain arbitrary noise. The example scanners and parsers in this document can only handle incorrect prefixes and rewinding when a checksum fails. Prefix noise may resemble a correct syntactic unit all the way down to the very last byte. The scanner must ultimately reject such noise and restart at its FSM's mandatory *Entry* vertex. When a checksum fails, the scanner assumes that

noise contained a valid header, so that the scanner mistakenly interpreted following bytes as payload and checksum. In this case, the scanner will push back all the misinterpreted characters, discard the deceptive header bytes, and restart. More general *error detection and correction* (EDAC) may become necessary in the future (also see *Algebraic Coding Theory* by Elwyn R. Berlekamp).

For instance, suppose we want to recognize the sequence 'ueTL' where 'u' and 'e' are the literal characters 'u' and 'e,' T is an arbitrary byte denoting the type of the payload data, and L is another byte whose value is the length of a following payload section, in bytes.

An FSM that can recognize such a sequence amidst arbitrary prefix bytes (special case of noise) appears below. The usual regular-expression notation obtains for negated character classes. That notation is a character sequence enclosed in square brackets with a leading caret. T and L are meta-variables that name the arbitrary characters denoted by the usual 'dot,' meaning "any character."

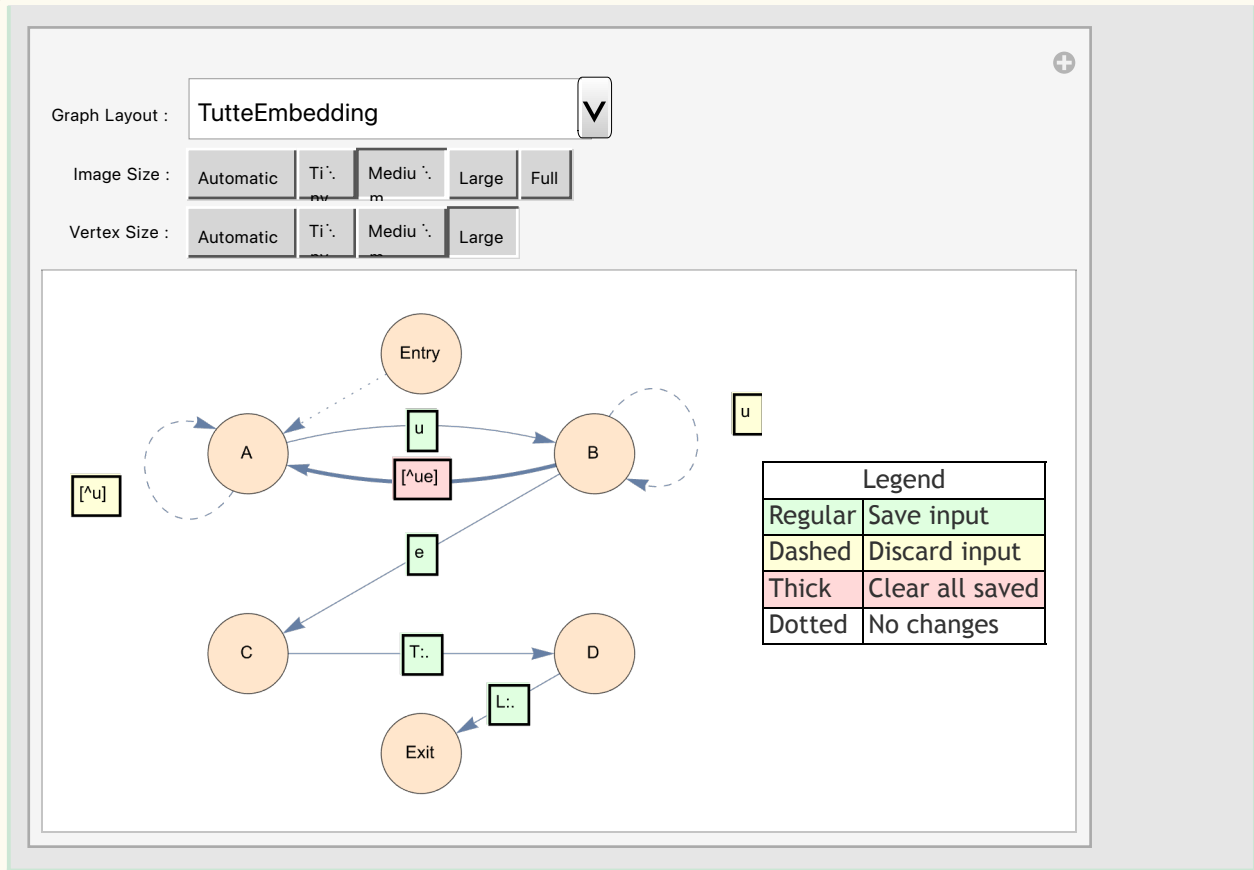
In[290]:=

```

Manipulate[
Module[{init, a, b, c, d, exit, fault,
  vertices, vertexLabel, edgeLabel, placed, label, labeled},
vertexLabel = v  $\mapsto$  Placed[v, Center];
edgeLabel = {e, bg}  $\mapsto$  Framed[e, Background  $\rightarrow$  bg];
vertices = {init, a, b, c, d, exit} = {"Entry", "A", "B", "C", "D", "Exit"};
Row[{Graph[
  Property[Style[#, LightOrange], VertexLabels  $\rightarrow$  vertexLabel[#]] & /@ vertices,
  {Style[init  $\rightarrow$  a, Dotted]},
  Property[Style[a  $\rightarrow$  a, Dashed],
  EdgeLabels  $\rightarrow$  edgeLabel["^u", LightYellow]],
  Property[a  $\rightarrow$  b, EdgeLabels  $\rightarrow$  edgeLabel["u", LightGreen]],
  Property[Style[b  $\rightarrow$  a, Thick], EdgeLabels  $\rightarrow$  edgeLabel["^ue", LightRed]],
  Property[Style[b  $\rightarrow$  b, Dashed], EdgeLabels  $\rightarrow$  edgeLabel["u", LightYellow]],
  Property[b  $\rightarrow$  c, EdgeLabels  $\rightarrow$  edgeLabel["e", LightGreen]],
  Property[c  $\rightarrow$  d, EdgeLabels  $\rightarrow$  edgeLabel["T:.", LightGreen]],
  Property[d  $\rightarrow$  exit, EdgeLabels  $\rightarrow$  edgeLabel["L:.", LightGreen]]},
VertexSize  $\rightarrow$  vs, ImageSize  $\rightarrow$  is, GraphLayout  $\rightarrow$  gl],
Grid[{{
  Style["Legend", "Text"], SpanFromLeft},
{Style["Regular", "Text"], Style["Save input", "Text"]},
{Style["Dashed", "Text"], Style["Discard input", "Text"]},
{Style["Thick", "Text"], Style["Clear all saved", "Text"]},
{Style["Dotted", "Text"], Style["No changes", "Text"]}},
Frame  $\rightarrow$  All, Alignment  $\rightarrow$  {Left, Left, {1, 1}  $\rightarrow$  Center},
Background  $\rightarrow$  {None, {White, LightGreen, LightYellow, LightRed, White}}}
]],
{{gl, "TutteEmbedding", "Graph Layout :"},
Sort@{Automatic, "RadialEmbedding", "SpectralEmbedding", "PlanarEmbedding",
"GridEmbedding", "LinearEmbedding", "BalloonEmbedding",
"TutteEmbedding", "StarEmbedding", "CircularEmbedding",
"DiscreteSpiralEmbedding", "SpiralEmbedding", "SpringEmbedding",
"SpringElectricalEmbedding", "HighDimensionalEmbedding"}},
{{is, Medium, "Image Size :"}, {Automatic, Tiny, Medium, Large, Full}},
{{vs, Large, "Vertex Size :"}, {Automatic, Tiny, Medium, Large}}]

```

Out[290]=



Definitions

- **buffer**: a sequential container for data, of a fixed *capacity* and *size*. In C, an array. In *Mathematica*, a *List*.
- **backwash**: a bogus type byte, length byte, payload bytes, and checksum bytes. They are discovered to be bogus when the checksum fails. The backwash is pushed back into the input and the scanner restarts.
- **length**: (Abbreviation: *len*) the number of validated items in a buffer. The meaning of *validated* depends on context. In a scanner, it means characters that have been recognized. In an input buffer, it means characters that have been transferred from an input stream into application memory.
- **capacity**: (Abbreviation: *cap*) the maximum number of items, valid or not, that a buffer may hold.
- **size**: the number of bytes or 8-bit characters in a capacity. This term is equivalent to *capacity* in this document because we only consider character data. This definition is useful for generalizations to other kinds of data to be scanned or parsed.
- **token**: in this work, we think of something like a “game token” that moves amongst states in an FSM. In the usual language of scanners, *token* means an indivisible item of input. We use the word “item” in this work to distinguish our tokens—like game tokens—from input items.
- **system**: the undefined context of hardware and software that uses the code explained in this document

- **interrupt service routine (ISR)**: an isolated block of code that runs from start to finish in response to hardware events, without interruption itself. An ISR has free access to system and application resources, including the data structures defined in this document, without concurrent contention from other code.
- **application**: whatever code is running, excluding ISRs, when an interruption occurs.
- **state**: Unfortunately, this word has multiple meanings in our context:
 - In general, a *state* is an association of values to all the variables in some universe of discourse. Here, this means the current association, i.e., assignment, of values to variables in the various data structures controlling the restartable i/o, scanners, and parsers.
 - the current vertex inhabited by the FSM representing the data to be recognized. Think of a game token that sits on one, and only one, vertex of the FSM graph at a time. The vertex on which the token sits is *the state of the FSM*. We write *vertex* to distinguish this meaning from the other meaning of *state*.

Stateful I/O

We achieve restartability with stateful I/O and with scanner routines that read from a scanner-state buffer and write back to the scanner-state buffer from an input buffer. Instances of the *Circular Buffer abstract data type* described below control the scanner state and input state and bound the memory used by the code.

INVARIANT: In our applications, the scanner-state circular buffer may contain only *valid prefixes*. In controlling the example FSM, for instance, the scanner state may contain nothing (written ϵ), 'u', 'ue', 'ueL', or 'ueLT'. The last possibility is a full syntactic unit, without its checksum, but is also considered a valid prefix.

■ Circular Buffer

The following design could trivially be made generic in the type of its contents. However, it is expressed non-generically on unsigned 8-bit characters (`uint8_t` in the C language) because that's the only type needed in the current application of parsing ASCII character data.

Though circular buffers seem like a trivial data type, we have found many bugs in practice, so we engineer a meticulous solution for the sake of high-assurance. A high-quality queue data type like Python's *deque* could be employed instead of this solution.

Design

■ Attributes

A circular buffer (CB) has

- a pointer to storage, *origin*

- a constant capacity *cap*, which is the maximum number of characters in *origin* (not necessarily the size in bytes, if this design were generalized to multibyte types of content)
- two indices into *origin*: *head* and *tail*
- a current length *len*, which is the number of valid characters stored in the CB and can be calculated on-the-fly

■ Semantics, Rules, and Restrictions

Characters are pushed to the tail and popped from the head. Pushed characters overwrite previous contents. Contents are not erased except by clearing the entire buffer or by pushing ***distinguished, invalid fill characters***.

Tail indexes the next cell available for pushing and not valid data. *Head* indexes the first valid data item when the CB is not empty. Pushing a character increments *tail*, modulo *cap*. The CB is empty when *tail* == *head*.

Notice that *len* can never be exactly *cap*. When *len* = *cap* - 1, the next *push* will fill the CB, but the CB will report that it's empty, i.e., that its length is zero. Thus, **this overall design cannot distinguish a full CB from an empty CB**. Distinguishing full from empty is the caller's responsibility.

To protect against improperly reading or testing invalid data, callers are advised to explicitly test for the empty condition of the CB and to initialize or clear the CB with a ***distinguished, invalid fill character*** that does not appear in normal use. The default distinguished, invalid fill character is zero.

This design is more efficient and less bug-prone than the alternatives, say of maintaining a separate length variable or of testing and setting a sentinel index for *head*. It is a well known solution. See <http://embedjournal.com/implementing-circular-buffer-embedded-c/> for example.

We assume efficient implementations of *mod* operations with respect to the capacity. Examples of efficient implementations are presented below.

This design silently overwrites old data on overflow. Alternative designs, not considered here, might discard new data or signal on overflow. These alternatives are rejected on grounds of ***complexity*** (more computer work) and ***complication*** (more programmer work).

■ Operations

□ Summary

- `reset [CB, head:Index, tail:Index] --> CB`
- `clear [CB] --> CB`
- `clear [CB, fillCharacter] --> CB`
- `push [CB, Character] --> CB`
- `pushIfDistinct [CB, Character] --> CB`
- `pushBack [CB, Character] --> CB`

- `pop [CB] --> Character`
- `backspace [CB] --> Character`
- `peek [CB] --> Character`
- `peek [CB, Index] --> Character`
- `peekLast [CB] --> Character`
- `peekLast [CB, Index] --> Character`
- `emptyQ [CB] --> Boolean`
- `fullQ [CB] --> Boolean`
- `almostFullQ [CB] --> Boolean`
- `len [CB] --> Index`

□ Details

The type *Index* is a non-negative integer. The type *Character* is an unsigned 8-bit integer, as noted above.

- `reset [CB, head:Index, tail:Index] --> CB`

Sets the head and tail of the CB to the given values. Used when the state of the CB is externally known.

Returns the modified CB for convenience in chained expressions.

- `clear [cb:CB] --> CB`

Fills *cb.origin* with zero, the default, invalid fill character, and resets *cb.head* and *cb.tail* to zero. The CB will report itself empty after a clear operation.

Doubles as initialization.

Returns the modified CB for convenience in chained expressions.

- `clear [cb:CB, fillCharacter:Character] --> CB`

Fills *cb.origin* with *fillCharacter* and resets *cb.head* and *cb.tail* to zero. The CB will report itself empty after a clear operation.

Doubles as initialization.

Returns the modified CB for convenience in chained expressions.

- `push [cb:CB, c:Character] --> CB`

Pushes character *c* (of type *Character*) onto the tail of *cb* (of type *CB*) incrementing *cb.tail* modulo *cb.cap*, even if the CB is full. This design silently overwrites the oldest bytes in the CB and may result in a newly empty CB containing arbitrary data.

Returns the modified CB for convenience in chained expressions.

- `pushIfDistinct [cb:CB, c:Character] --> CB`

Push character *c* (of type *Character*) onto the tail of *cb* (of type *CB*) if and only if the last character in *cb*, namely the character at position $(cb.tail - 1) \% cb.cap$, is not *c*, **even if the CB is full or empty**.

The empty condition is significant because *pushIfDistinct* can test any character, not necessarily the one intended by the programmer. This operation may test the invalid fill character of *cb* or, in fact, any character left over in the CB from prior operations.

Callers are advised to explicitly test for an empty CB and to initialize or clear the CB with a distinguished, invalid fill character. The default distinguished, invalid fill character is zero.

Returns the modified CB for convenience in chained expressions.

■ `pushBack [cb:CB, c:Character] --> CB`

Pushes character *c* (of type *Character*) onto the head of *cb* (of type *CB*) decrementing *cb.head* modulo *cb.cap*, **even if the CB is full**. This design silently overwrites prior bytes in the CB.

Returns the modified CB for convenience in chained expressions.

■ `pop [cb:CB] --> Character`

Returns the character at the head of *cb* and advances *cb.head* modulo *cb.cap*, **even if the CB is empty**.

This overall design does not distinguish an empty CB from a full CB, so *pop* on an empty CB may return any character: the fill character or any character that may be left over from prior operations.

Callers are advised to explicitly test for an empty CB and to initialize or clear the CB with a distinguished, invalid fill character. The default distinguished, invalid fill character is zero.

■ `backspace [cb:CB] --> Character`

Returns the character just prior to the tail, at position $(cb.tail - 1) \% cb.cap$, and decrements *cb.tail* modulo *cb.cap*, **even if the CB is empty**.

This overall design does not distinguish an empty CB from a full CB, so *backspace* on an empty CB may return any character: the fill character or any character that may be left over from prior operations.

Callers are advised to explicitly test for an empty CB and to initialize or clear the CB with a distinguished, invalid fill character. The default distinguished, invalid fill character is zero.

■ `peek [cb:CB] --> Character`

Returns the character at the head of *cb* without advancing *cb.head*, **even if the CB is empty**.

This overall design does not distinguish an empty CB from a full CB, so *peek* on an empty CB may return any character: the fill character or any character that may be left over from prior operations.

Callers are advised to explicitly test for an empty CB and to initialize or clear the CB with a distinguished, invalid fill character. The default distinguished, invalid fill character is zero.

■ `peek [cb:CB, i:Index] --> c`

Returns the character at position $(cb.head + i) \% cb.cap$, **even if the CB is empty**.

This overall design does not distinguish an empty CB from a full CB, so *peek* on an empty CB may return any character: the fill character or any character that may be left over from prior operations.

Callers are advised to explicitly test for an empty CB and to initialize or clear the CB with a distinguished, invalid fill character. The default distinguished, invalid fill character is zero.

■ `peekLast [cb:CB] --> Character`

Returns the character just before the tail of *cb*. that is, at position $(cb.tail - 1) \% cb.cap$, **even if the CB is empty**.

This overall design does not distinguish an empty CB from a full CB, so *peekLast* on an empty CB may return any character: the fill character or any character that may be left over from prior operations.

Callers are advised to explicitly test for an empty CB and to initialize or clear the CB with a distinguished, invalid fill character. The default distinguished, invalid fill character is zero.

■ `peekLast [cb:CB, i:Index] --> Character`

Returns the character at position $(cb.tail - i) \% cb.cap$, **even if the CB is empty**.

This overall design does not distinguish an empty CB from a full CB, so *peekLast* on an empty CB may return any character: the fill character or any character that may be left over from prior operations.

Callers are advised to explicitly test for an empty CB and to initialize or clear the CB with a distinguished, invalid fill character. The default distinguished, invalid fill character is zero.

■ `emptyQ [cb:CB] --> Boolean`

Equivalent to testing that $len[cb] == 0$. **Note this is also true when the CB is full**. Distinguishing full from empty is the caller's responsibility.

■ `fullQ [cb:CB] --> Boolean`

Synonym for *emptyQ*. Equivalent to testing that $len[cb] == 0$. **Note this is also true when the CB is empty**. Distinguishing full from empty is the caller's responsibility.

■ `almostFullQ [cb:CB] --> Boolean`

Equivalent to testing that $len[cb] == cb.cap - 1$. The next push will render the CB full (and empty).

■ `len [cb:CB] --> non-negative integer`

Returns $(cb.tail - cb.head) \% cb.cap$. **This reports zero when the CB is empty or full**. Distinguishing full from empty is the caller's responsibility.

Reference Implementation

The following code is as close to C code as Mathematica allows us to write. Its purpose is to serve as an executable design for C code. The writing of high-assurance C code should be straightforward from this design.

All indexing expressions below, where indices appear between doubled square brackets, as in `p[[i]]`, are written with the index +1. Transcribe those expressions into C without the +1. All other arithmetic on indices is identical to what we would write in C. We adopt this technique to isolate the differences between Mathematica (1-based indexing) and C (0-based indexing) to that one usage.

Mathematica is semi-functional: values are immutable, but variables can be assigned. This has four ramifications for our reference implementation:

1. We cannot modify values by side-effect. We can make modified copies, then assign the modified copies to variables.
2. Function parameters are not variables. Inside functions, we must explicitly make temporary variables in *Module* forms and copy the values of parameters.
3. In the *pop* and *backspace* operations, we return pairs of the modified (copies of) the CB and the character popped or backspaced over.
4. In scanners and parsers, we make frequent use of multiple-return values in an obvious pattern similar to that for item 3, above.

■ Support Functions

We do a cheap imitation of modular arithmetic by adding the modulus to a sum or difference and then subtracting it if and only if necessary. This is risky if the inputs are large, but note that gcc generates efficient code for this case <http://coliru.stacked-crooked.com/a/27397a28f423c3eb>.

It's easy to substitute other implementations if they're more efficient. Examples include *mod* computations with respect to powers of two, for which the C compiler can generate efficient code.

□ cheapMod [a, modulus] ~~> a % modulus

In[291]:=

```
ClearAll[cheapMod];
cheapMod[a_, modulus_] :=
  Module[{j = a},
    While[j < 0, j += modulus];
    While[j ≥ modulus, j -= modulus];
    (*return*)j];
```

□ cheapAddMod [a, b, modulus] ~~> (a + b) % modulus

In[293]:=

```
ClearAll[cheapAddMod];
cheapAddMod[a_, b_, modulus_] :=
  (*return*)cheapMod[a + b, modulus];
```

□ cheapIncMod [a, modulus] ~~> (a + 1) % modulus

In[295]:=

```
ClearAll[cheapIncMod];
cheapIncMod[a_, modulus_] := cheapAddMod[a, 1, modulus];
```

▣ cheapDecMod [a, modulus] $\rightsquigarrow (a - 1) \% \text{modulus}$

In[297]:=

```
ClearAll[cheapDecMod];
cheapDecMod[a_, modulus_] := cheapAddMod[a, -1, modulus];
```

■ Circular-Buffer Operations

Represent a CB with a *struct* (Mathematica's equivalent of a struct is an *Association*: a set of name-value pairs inside `<| |>` brackets.

The type of a character is an Integer between 0 and 255 inclusive both ends, to better match the C language.

▣ Reset

In[299]:=

```
ClearAll[CBreset];
CBreset[cb_, head_, tail_] :=
  Module[{temp = cb},
    temp["head"] = head;
    temp["tail"] = tail;
    (*return*)temp];
```

▣ Clear

In[301]:=

```
ClearAll[isChar];
isChar[c_] := IntegerQ[c] && (c ≥ 0) && (c < 256);
```

In[303]:=

```
ClearAll[CBclear];
CBclear[cb_] :=
  Module[{temp = cb},
    temp["origin"] = ConstantArray[0, cb["cap"]];
    (*return*)CBreset[temp, 0, 0];
  (*overload*)
  CBclear[cb_, fillCharacter_] :=
    Module[{temp = cb},
      temp["origin"] = ConstantArray[fillCharacter, cb["cap"]];
      (*return*)CBreset[temp, 0, 0];
```

▣ Constructor

We do not need this in C code. It's included here for convenience in testing below.

We require the caller to supply the *origin* buffer, that is, the storage that will house the contents of the circular buffer.

In[306]:=

```
ClearAll[makeCB];
makeCB[
  cap_Integer /; ((cap > 0) && (cap ≤ 512)),
  origin_List /; (Length[origin] === cap) :=
  <|"cap" → cap, "origin" → origin, "head" → 0, "tail" → 0|>;
```

In[308]:=

```
ClearAll[makeTestCB];
makeTestCB[cap_Integer /; ((cap > 0) && (cap ≤ 512))] :=
  makeCB[cap, ConstantArray[0, cap]];
```

□ Push

In[310]:=

```
ClearAll[CBpush];
CBpush[cb_, c_Integer /; isChar[c]] :=
  Module[{
    temp = cb,
    originTemp = cb["origin"], (*lvalue temps*)
    tail = cb["tail"]},
    With[{newTail = cheapIncMod[tail, cb["cap"]]},
      originTemp[[1+tail]] = c; (* Take out the 1+ in C. *)
      temp["origin"] = originTemp;
      temp["tail"] = newTail;
      (*return*) temp]];
```

□ PushIfDistinct

In[312]:=

```
ClearAll[CBpushIfDistinct];
CBpushIfDistinct[cb_, c_Integer /; isChar[c]] :=
  (*Mathematica If[a,b,c] is like C a?b:c*)
  (*return*) If[c != CBpeekLast[cb], CBpush[cb, c], cb];
```

□ PushBack

In[314]:=

```
ClearAll[CBpushBack];
CBpushBack[cb_, c_Integer /; isChar[c]] :=
Module[{
  temp = cb,
  originTemp = cb["origin"], (*lvalue temps*)
  head = cb["head"]},
With[{newHead = cheapDecMod[head, cb["cap"]]},
  originTemp[[1+newHead]] = c; (* Take out the 1+ in C. *)
  temp["origin"] = originTemp;
  temp["head"] = newHead;
  (*return*) temp]];
```

□ Pop

Mathematica (semi-functional) forces us to construct a modified copy of the CB. We return a pair, as a *Mathematica List* in curly braces, of the modified copy and the popped character data.

In[316]:=

```
ClearAll[CBpop];
CBpop[cb_] :=
Module[{
  temp = cb,
  result = CBpeek[cb]},
temp["head"] = cheapIncMod[cb["head"], cb["cap"]];
(*return*) {temp, result}];
```

□ Backspace

Mathematica (semi-functional) forces us to construct a modified copy of the CB. We return a pair, as a *Mathematica List* in curly braces, of the modified copy and the backspaced character data.

In[318]:=

```
ClearAll[CBbackspace];
CBbackspace[cb_] :=
Module[{
  temp = cb,
  result = CBpeekLast[cb]},
temp["tail"] = cheapDecMod[cb["tail"], cb["cap"]];
(*return*) {temp, result}];
```

□ Peek

In[320]:=

```
ClearAll[CBpeek];
CBpeek[cb_] := cb["origin"][[1 + cb["head"]]]; (*take out 1+ in C*)
CBpeek[cb_, i_] :=
  (*return*) cb["origin"][[1 + cheapAddMod[cb["head"], i, cb["cap"]]]];
  (*take out the 1+ in C*)
```

□ PeekLast

In[323]:=

```
ClearAll[CBpeekLast];
CBpeekLast[cb_] := CBpeekLast[cb, 1];
CBpeekLast[cb_, i_] :=
  (*return*) cb["origin"][[1 + cheapAddMod[cb["tail"], -i, cb["cap"]]]];
  (*take out the 1+ in C*)
```

□ EmptyQ

In[326]:=

```
ClearAll[CBemptyQ];
CBemptyQ[cb_] := (*return*) cb["head"] === cb["tail"];
```

□ FullQ

In[328]:=

```
ClearAll[CBfullQ];
CBfullQ = (*synonym*) CBemptyQ;
```

□ AlmostFullQ

In[330]:=

```
ClearAll[CBalmostFullQ];
CBalmostFullQ[cb_] := (*return*)
  cb["head"] === cheapIncMod[cb["tail"], cb["cap"]];
```

□ Len

In[332]:=

```
ClearAll[CBlen];
CBlen[cb_] := (*return*) cheapAddMod[cb["tail"], -cb["head"], cb["cap"]];
```

Interactive Testing

In[334]:=

```
ClearAll[insertVectorAsColumn, insertColumns, vectorToColumn];
```

```

insertVectorAsColumn[column_, matrix_, position_: 1] :=
  MapThread[Insert[#2, #1, position] &, {column, matrix}];
insertColumns[m1_, m2_] := MapThread[Join, {m1, m2}];
vectorToColumn[v_] := Partition[v, 1];
ClearAll[CBdisplay];
CBdisplay[cb_] :=
  With[{n = " "},
    Module[{cap = cb["cap"], head = cb["head"],
      tail = cb["tail"], headDpy, tailDpy, indices},
      headDpy = ConstantArray[n, cap];
      tailDpy = ConstantArray[n, cap];
      headDpy[[1 + head]] = "^";
      tailDpy[[1 + tail]] = "^";
      indices = Range[0, cap - 1];
      Row[{
        Grid[vectorToColumn[{"index:", "origin:", "head:", "tail:"}],
          Alignment → Left],
        Grid[{indices, cb["origin"], headDpy, tailDpy},
          Alignment → Right, Frame → All, ItemSize → All,
          Background → Evaluate[{None, None, (
            {2, # + 1} →
              If[cb["head"] < cb["tail"] && cb["head"] ≤ # && # < cb["tail"],
                LightYellow,
                If[cb["head"] > cb["tail"] && (cb["head"] ≤ # || # < cb["tail"]),
                  LightYellow,
                  LightGray]} & /@ Range[0, cb["cap"] - 1]}]]]]];
ClearAll[capitals, uncials, letters, digits, hexits, punctuation,
  alphamerics, graphicals, controls, blanks, spaces, printables];
capitals = Range[16^41, 16^5A];
uncials = Range[16^61, 16^7A];
letters = capitals~Join~uncials;
digits = Range[16^30, 16^39];
hexits = digits~Join~capitals[[1 ;; 6]]~Join~uncials[[1 ;; 6]];
punctuation = Range[16^21, 16^2F]~Join~
  {16^40}~Join~
  Range[16^5B, 16^5F]~Join~
  {16^60}~Join~
  Range[16^7B, 16^7E];
alphamerics = letters~Join~digits;
graphicals = alphamerics~Join~punctuation;
spaces = Range[16^9, 16^D]~Join~{16^20};
printables = graphicals~Join~spaces;
controls = Range[16^0, 16^1F]~Join~{16^7F};
blanks = {16^9, 16^20};

```

```

DynamicModule[{cPopped = 0, cBackspaced = 0, cb = makeTestCB[16]},
Column[
  {Row[{
    Button["Push Random Char", cb = CBpush[cb, RandomChoice[graphicals]]],
    Button["Pushback Random", cb = CBpushBack[cb, RandomChoice[graphicals]]],
    Button["Push Indistinct", cb = CBpushIfDistinct[cb, CBpeekLast[cb]]],
    Button["Backspace", {cb, cBackspaced} = CBbackspace[cb]],
    Button["Pop", {cb, cPopped} = CBpop[cb]],
    Button["Clear", cb = CBclear[cb]]}],
  Dynamic[
    Grid[{
      {"len:", CBlen[cb], "peek:", CBpeek[cb],
        "last:", CBpeekLast[cb], "empty?", CBemptyQ[cb]},
      {"last popped:", cPopped, "last backspaced:", cBackspaced,
        "", "", "almost full?", CBalmostFullQ[cb]}},
      Frame → All, Alignment → Right]],
    Dynamic[CBdisplay[cb]]}]]

```

Out[353]=

Push Random Char		Pushback Random				Push Indistinct				Backspace		Pop	Clear			
len: 0		peek: 0				last: 0		empty?		True						
last popped: 0		last backspaced: 0						almost full?		False						
index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
origin:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
head:	^															
tail:	^															

A Restartable Scanner

The system repeatedly interrupts the application with one or more new input bytes from an input stream, say a peripheral device. More completely, the interrupt service routine, ISR, receives (a pointer to) the input buffer and a length, the number of characters transferred to the system from the peripheral. The system does not know the capacity of the caller-supplied input buffer: it's the responsibility of the caller (the application) to ensure there is enough room so that input buffers are not overflowed.

Algorithm

The scanner models its current scanner state with a circular buffer called *scannerState*, another buffer called *provisionalPayload*, and wraps the input with another circular buffer called *inputState*. NOTE: in the C implementation, we do not use the circular-buffer abstraction for provisional payload or for the input state, but rather simple *push*, *pop*, and *backup* routines over a zero-based array.

INVARIANT: The scanner state only ever contains valid characters, either newly validated or already validated in a prior pass.

On reentry, the scanner restarts its FSM at its unique *Entry* vertex and the scanner saves the current head index of the scanner state. The scanner steps the FSM by feeding it validated characters popped from the scanner state until the scanner state is exhausted. If there are any such characters, the FSM eventually reaches the vertex achieved in a prior pass. The scanner then resets the head of the scanner state to its prior value because the scanner is assured, by the invariant above, that all popped characters were valid. From then on, the scanner steps the FSM with characters from the input state.

The scanner maintains two metrics: *discarded* and *saved*, in a struct called *metricState*. The *discarded* metric counts the number of characters discarded while taking transitions along dashed edges in the diagram: edges with the “Discard input” label. The *saved* metric counts characters saved from the input state to the scanner state while taking regular edges with the “Save input” label. While taking thick edges with the “Clear all saved” label, *discarded* is incremented by the length of the scanner state and the scanner state is cleared; *saved* is set to zero. “Washback” edges are described later. They do not appear in the FSM above, which only recognizes headers.

The scanner and FSM maintain a dictionary called *matches* for edges labeled with meta-variables *L* and *T*. This dictionary keeps the character(s) recognized under the corresponding meta-variable.

As usual when designing C code with *Mathematica*, we make temporary variables in a *Module* for mutable structures, and initialize those temporaries from parameters. The reason is that parameters are not variables in *Mathematica*, thus parameters cannot be updated. The corresponding routines in C will simply modify *struct* contents in-place.

FsmStep is a foldable function, meaning that the return value has the same type as the first input. This is important for high-assurance software: foldables can be tested independently of their callers (see *A Tutorial on the Universality and Expressiveness of Fold* by Paul Graham and this author’s *Kalman Folding*).

The return type of *fsmStep*, same as the type of the first parameter, is a Mathematica *List* in curly braces, of the prior vertex, an action command, a parameter for the action, and the current *matches* dictionary. The input action and its parameter is ignored. The returned actions are “save,” “discard,” “clear,” “washback,” or “no action;” it’s up to the caller to decide whether to perform them. The following is a direct transcription of the diagram in the first part of this document.

NOTE: this executable design (or reference implementation) contains exception-throwing to assert invariants. Such constructs must not appear in embedded builds.

A Stepping FSM

The following is a straightforward transcription of the diagram at the beginning of this document.

In[354]:=

```
ClearAll[fsmStep];
With[{(*constants*)
```

```

uCode = First@ToCharacterCode["u"],
eCode = First@ToCharacterCode["e"]},
fsmStep[{
  oldVertex_,
  ignoredActionParam_,
  ignoredActionParamParam_,
  matchesParam_},
character_] :=
Module[(*local variables*)
  matches = matchesParam,
  newVertex,
  action,
  actionArgument},

{newVertex, action, actionArgument} =

Switch[oldVertex,

  "Entry",
  {"A", "no action", "no arg"},

  "A",
  Switch[character,
    uCode, {"B", "save", "validated"},
    default_, {"A", "discard", "no arg"}],

  "B",
  Switch[character,
    uCode, {"B", "discard", "no arg"},
    eCode, {"C", "save", "validated"},
    default_, {"A", "clear", "no arg"}],

  "C", (matches["T"] = character; {"D", "save", "validated"}),

  "D", (matches["L"] = character; {"Exit", "save", "validated"}),

  "Exit",
  {"Exit", "no action", "no arg"},

  default_,
  {0, Throw["Illegal fsm vertex: " <> ToString@oldVertex]}
];

(*Print[<|

```

```

    "oldVertex"→oldVertex,
    "action"→action,
    "arg"→actionArgument,
    "newVertex"→newVertex,
    "matches"→matchesParam,
    "new matches"→matches]>];*)

(*return*)
{newVertex, action, actionArgument, matches}

]];

```

A Restartable Scanner

Scanner simulates a fold over *fsmStep*. *Scanner* is responsible only for buffer management. It gets action commands from *fsmStep*, but does different things when characters come from the scanner state, provisional payload, or from the input state. It does not know details of the FSM, knowing only the *Entry* and *Exit* vertices that every FSM must have. This scanner is generic for any FSM that has the structure {header, provisional payload, checksum}.

■ Support Routines

The FSM will give the scanner action commands. Some action commands involve moving characters (at least virtually) from one buffer to another. The routines *washback* and *transferAll* support these commands.

In[356]:=

```

ClearAll[washBack];
washBack[destBuf_, srcBuf_, n_] :=
  Module[{c, i = n, dest = destBuf, src = srcBuf},
    While[Not[CBemptyQ[src]] && (i > 0),
      i = i - 1;
      {src, c} = CBbackspace[src];
      dest = CBpushBack[dest, c];
      {dest, src}];
ClearAll[transferAll];
transferAll[destBuf_, srcBuf_] :=
  Module[{c, dest = destBuf, src = srcBuf},
    While[Not[CBemptyQ[src]],
      {src, c} = CBpop[src];
      dest = CBpush[dest, c];
      {dest, src}];

```

In[360]:=

```
Module[{src = makeTestCB[16], dest = makeTestCB[16]},
  src = Fold[CBpush[#1, #2] &, src, Range[7]];
  {dest, src} = washBack[dest, src, 4];
  Column[{
    "***** dest *****",
    CBdisplay[dest],
    "***** src *****",
    CBdisplay[src]}]]
```

Out[360]=

```
***** dest *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin:  0  0  0  0  0  0  0  0  0  0  0  0  4  5  6  7
head:    ^
tail:    ^

***** src *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin:  1  2  3  4  5  6  7  0  0  0  0  0  0  0  0  0
head:    ^
tail:    ^
```

In[361]:=

```
Module[{src = makeTestCB[16], dest = makeTestCB[16]},
  src = Fold[CBpush[#1, #2] &, src, Range[7]];
  {dest, src} = transferAll[dest, src];
  Column[{
    "***** dest *****",
    CBdisplay[dest],
    "***** src *****",
    CBdisplay[src]}]]
```

Out[361]=

```
***** dest *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin:  1  2  3  4  5  6  7  0  0  0  0  0  0  0  0  0
head:    ^
tail:    ^

***** src *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin:  1  2  3  4  5  6  7  0  0  0  0  0  0  0  0  0
head:    ^
tail:    ^
```

■ **Scanner [scannerStateCB, provisionalPayloadCB, inputStateCB, fsmStepper]**
→ scannerStateCB, provisionalPayloadCB, inputStateCB, metrics, matches

Three CBs are input and modified by side effect.

Not a foldable form because characters may wash back from provisional payload to the input. Foldable requires the second input to be a sequential source that's completely consumed.

In[362]:=

```
ClearAll[scanner];
scanner[
  scannerStateParam_,
  provisionalPayloadParam_,
  inputStateParam_,
  fsmStepFunction_] :=
Module[(*local variables*)
  (*shallow copies*)
  ss = scannerStateParam,
  ps = provisionalPayloadParam,
  is = inputStateParam,
  (*for output*)
  metrics = <|"discarded" → 0, "saved" → 0|>,
  matches = <|>,
  (*for stepping the FSM*)
  oldHead, (*Save original scanner-
    state head because pops of valid characters may occur.*)
  vertex,
  character,
  action, (*ignored at first*)
  actionArgument(*ignored at first*)},
(* Advance the FSM from Entry to its first actionable state. *)
{vertex, action, actionArgument, matches} =
  fsmStepFunction[{"Entry", Null, Null, matches}, Null];
(* Step the FSM with validated characters from scanner state. *)
oldHead = ss["head"];
While[Not[CBemptyQ[ss]],
  (* No need to check for Exit and Fault because of the invariant
    that the scanner state contains only valid characters. *)
  {ss, character} = CBpop[ss];
  {vertex, action, actionArgument, matches} =
    fsmStepFunction[{vertex, action, actionArgument, matches}, character];
  Switch[action,
    "save", Null, (*do nothing;
      characters in scanner state have already been validated*)
```

```

"clear", Throw["1: Invalid character "<>
  ToString@character<>" in scanner state"], (*should never happen*)
"discard", Throw["2: Invalid character "<>
  ToString@character<>" in scanner state"], (*should never happen*)
"no action", Throw["5: Invalid character "<>
  ToString@character<>" in scanner state"], (*should never happen*)
default_, Throw["3: Invalid action "<>
  ToString@action<>" returned by fsm step on character "<>
  ToString@character] (*should never happen*)
](*switch action*)
]; (*while scanning prior scanner state*)
(* Reset head of scanner state to old head
because pops of valid characters may have occurred. *)
ss = CBreset[ss, oldHead, ss["tail"]];
While[Not[CBemptyQ[is]],
  (*Take characters from input state*)
  {is, character} = CBpop[is];

  {vertex, action, actionArgument, matches} =
    fsmStepFunction[
      {vertex, action, actionArgument, matches},
      character];

  Switch[action,

    "save",
    Switch[actionArgument,
      "validated", (metrics["saved"] ++;
        ss = CBpush[ss, character];),
      "provisional", (ps = CBpush[ps, character]),
      default_,
      Throw["6: Invalid action argument "<>ToString[actionArgument]]],

    "clear",
    (metrics["discarded"] += CBleng[ss];
      metrics["saved"] = 0;
      ss = CBclear[ss]),

    "discard",
    (metrics["discarded"] ++),

    "washback",
    With[{count = actionArgument},
      If[Not[NumberQ[count]] || count < 0,

```

```

        Throw["6: invalid washback count " <> ToString[count]];
        {is, ps} = washBack[is, ps, count]],

    "no action",
    Null,

    default_,
    Throw["4: Invalid action " <> ToString[action <>
        " returned by fsm step on character " <> ToString[character]

]; (*switch action*)

If[vertex === "Exit",
    ({ss, ps} = transferAll[ss, ps]); Break[]];

If[vertex === "Fault",
    (ss = CBclear[ss];
    Break[])];

];
(*while scanning input and transferring chacters to scanner state*)

(*return*){ss, ps, is, metrics, matches}
];

```

Testing FSM and Scanner

■ Support Routines

In[364]:=

```

ClearAll[initializedCB];
initializedCB[s_String] :=
    Fold[CBpush[#1, #2] &, makeTestCB[16], ToCharacterCode[s]];

```

In[366]:=

```
ClearAll[testScanner];
testScanner[scannerChars_String, inputChars_String, fsmStepper_ : fsmStep] :=
Module[{
  ss = initializedCB[scannerChars],
  ps = initializedCB[""],
  is = initializedCB[inputChars],
  metrics,
  matches},
{ss, ps, is, metrics, matches} = scanner[ss, ps, is, fsmStepper]];
```

In[368]:=

```
successes = {}; failures = {};
```

In[369]:=

```
ClearAll[expect];
expect[expected_, actual_] :=
If[expected === actual,
  (AppendTo[successes, actual]; "PASS"),
  (AppendTo[failures, "Expected " <>
    ToString[expected] <> " is not actual " <> ToString[actual] <> "."];
    "FAIL")];
```

■ A Single, Invalid Input Character

In[371]:=

```
expect[testScanner["", "j"],
  {<|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {106, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 1, "tail" → 1|>, <|"discarded" → 1, "saved" → 0|>, <||>}]
```

Out[371]:=

```
PASS
```


■ A Single, Valid Input Character

In[372]:=

```
expect[testScanner["", "u"],
  {<|"cap" → 16, "origin" → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 1|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 1, "tail" → 1|>, <|"discarded" → 0, "saved" → 1|>, <|>}]
```

Out[372]=

PASS

■ A Complete, Valid Sequence

In[373]:=

```
expect[testScanner["", "ueJK"],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 4, "tail" → 4|>, <|"discarded" → 0, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

Out[373]=

PASS

■ Incomplete Scanner State

In[374]:=

```
expect[testScanner["u", ""],
  {<|"cap" → 16, "origin" → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 1|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"discarded" → 0, "saved" → 0|>, <|>}]
```

Out[374]=

PASS

■ Stuttering

In[375]:=

```
expect[testScanner["u", "uuuuuuueJK"],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 117, 117, 117, 117, 117, 117,
      101, 74, 75, 0, 0, 0, 0, 0, 0}, "head" → 10, "tail" → 10|>,
    <|"discarded" → 7, "saved" → 3|>, <|"T" → 74, "L" → 75|>}]
```

Out[375]=

PASS

■ Scanner State with Ignored Valid Characters

In[376]:=

```
expect[testScanner["ue", "xyzabcdef"],
  {<|"cap" → 16, "origin" → {117, 101, 120, 121, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {120, 121, 122, 97, 98, 99, 100,
      101, 102, 0, 0, 0, 0, 0, 0, 0}, "head" → 2, "tail" → 9|>,
    <|"discarded" → 0, "saved" → 2|>, <|"T" → 120, "L" → 121|>}]
```

Out[376]=

PASS

In[377]:=

```
expect[testScanner["ueX", "YZABCDEF"],
  {<|"cap" → 16, "origin" → {117, 101, 88, 89, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {89, 90, 65, 66, 67, 68, 69, 70, 0, 0, 0, 0, 0, 0, 0},
    "head" → 1, "tail" → 8|>, <|"discarded" → 0, "saved" → 1|>, <|"T" → 88, "L" → 89|>}]
```

Out[377]=

PASS

■ A Little Junk in the Input

In[378]:=

```
expect[testScanner["u", "asdfueJK"],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {97, 115, 100, 102, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0},
    "head" → 8, "tail" → 8|>, <|"discarded" → 4, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

Out[378]=

PASS

In[379]:=

```
expect[testScanner["u", "XueJK"],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {88, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 5, "tail" → 5|>, <|"discarded" → 1, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

Out[379]=

PASS

In[380]:=

```
expect[testScanner["u", "uXueJK"],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 88, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 6, "tail" → 6|>, <|"discarded" → 2, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

Out[380]=

PASS

In[381]:=

```
expect[testScanner["u", "uXXueJK"],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 88, 88, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0},
    "head" → 7, "tail" → 7|>, <|"discarded" → 3, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

Out[381]=

PASS

In[382]:=

```
expect[testScanner["u", "uXXuueJK"],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 88, 88, 117, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0},
    "head" → 8, "tail" → 8|>, <|"discarded" → 4, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

Out[382]=

PASS

■ Lots of Junk in the Input

In[383]:=

```
expect[testScanner["u", "uuuuuuuXXXuueJK"],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 117, 117, 117, 117, 117, 117, 88,
    88, 88, 117, 117, 101, 74, 75, 0}, "head" → 15, "tail" → 15|>,
    <|"discarded" → 11, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

Out[383]=

PASS

■ All Bets Are Off

Input-state circular buffer is full, therefore it is also empty. Caller has violated the contract by not ensuring valid contents, so caller should not expect consistent results

In[384]:=

```
expect[testScanner["u", "uuuuuuuXXXXuueJK"],
  {<|"cap" → 16, "origin" → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 1|>,
    <|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"cap" → 16, "origin" →
    {117, 117, 117, 117, 117, 117, 117, 117, 88, 88, 88, 88, 117, 117, 101, 74, 75},
    "head" → 0, "tail" → 0|>, <|"discarded" → 0, "saved" → 0|>, <|>|>}]
```

Out[384]=

PASS

■ Results

In[385]:=

```
<|"success count" → Length@successes, "failure count" → Length@failures,
  "successes" → successes, "failures" → failures|>
```

Out[385]=

```
<|success count → 14, failure count → 0,
  successes → {{<|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {106, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 1, tail → 1|>, <|discarded → 1, saved → 0|>, <| |>},
  {<|cap → 16, origin → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 1|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 1, tail → 1|>, <|discarded → 0, saved → 1|>, <| |>},
  {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 4, tail → 4|>, <|discarded → 0, saved → 4|>, <|T → 74, L → 75|>},
  {<|cap → 16, origin → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 1|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|discarded → 0, saved → 0|>, <| |>},
  {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0,
    tail → 4|>, <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|cap → 16,
    origin → {117, 117, 117, 117, 117, 117, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0},
    head → 10, tail → 10|>, <|discarded → 7, saved → 3|>, <|T → 74, L → 75|>},
  {<|cap → 16, origin → {117, 101, 120, 121, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {120, 121, 122, 97, 98, 99, 100, 101, 102, 0, 0, 0, 0, 0, 0},
    head → 2, tail → 9|>, <|discarded → 0, saved → 2|>, <|T → 120, L → 121|>},
  {<|cap → 16, origin → {117, 101, 88, 89, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {89, 90, 65, 66, 67, 68, 69, 70, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 1, tail → 8|>, <|discarded → 0, saved → 1|>, <|T → 88, L → 89|>},
  {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```

    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {97, 115, 100, 102, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0},
    head → 8, tail → 8|>, <|discarded → 4, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {88, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 5, tail → 5|>, <|discarded → 1, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 88, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 6, tail → 6|>, <|discarded → 2, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 88, 88, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0},
    head → 7, tail → 7|>, <|discarded → 3, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 88, 88, 117, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0},
    head → 8, tail → 8|>, <|discarded → 4, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0,
    tail → 4|>, <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|cap → 16, origin →
    {117, 117, 117, 117, 117, 117, 117, 88, 88, 88, 117, 117, 101, 74, 75, 0},
    head → 15, tail → 15|>, <|discarded → 11, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0,
    tail → 1|>, <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|cap → 16, origin →
    {117, 117, 117, 117, 117, 117, 88, 88, 88, 88, 117, 117, 101, 74, 75},
    head → 0, tail → 0|>, <|discarded → 0, saved → 0|>, <|>}}}, failures → {}|>

```

Scanning More

The FSM above just accepts a header, but the header contains a length byte, *L*, and a type byte, *T*. The header tells us whether there is more data downstream. The length byte *L* gives us the entire length of validated payload characters, excluding the header **ueTL**.

The downstream data consists of one or more *fields*. Each field contains a *relative index byte* followed by content bytes. The relative index byte of a field equals the length of the field, including the relative

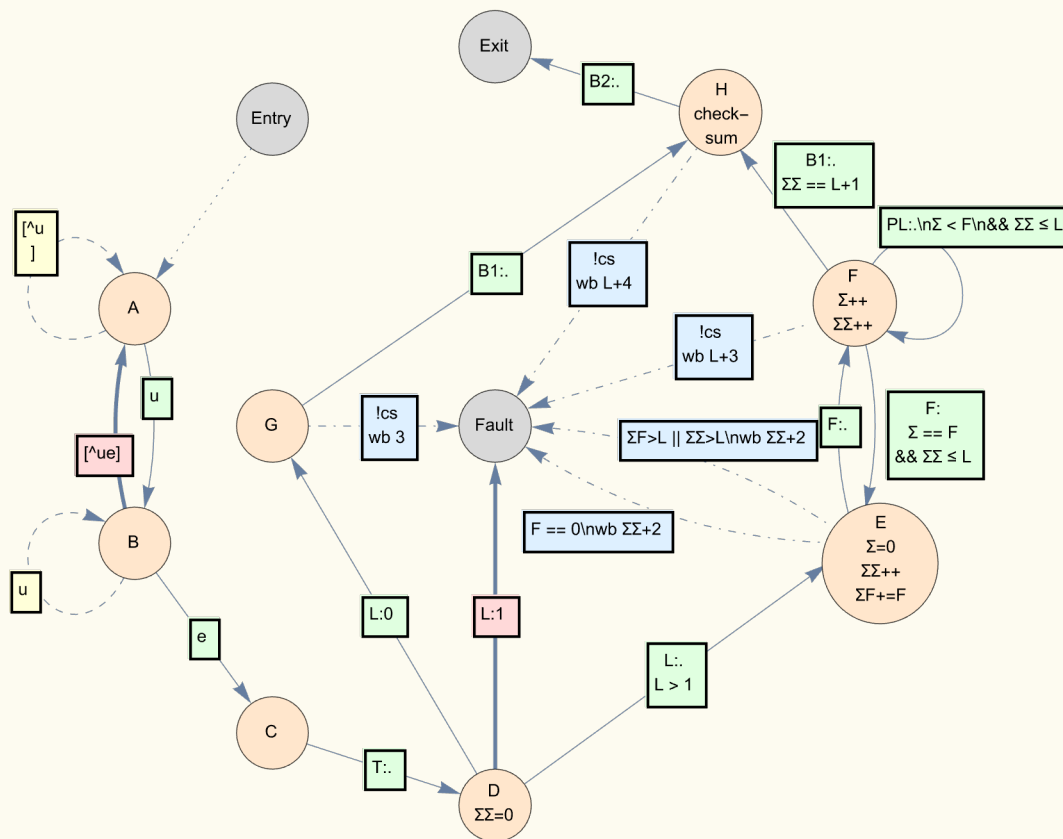
index byte, itself. Therefore, INVARIANTS: F cannot be zero; L cannot be one; the sum of all relative index bytes F must equal the total length byte, L .

New FSM

The full FSM maintains running variables for the checksum, initialized on entry. It keeps three more variables in the FSM --- Σ , $\Sigma\Sigma$, and ΣF --- to track the lengths of each field, of all the fields, and the sum of all F bytes. Save validated bytes of the header to scanner state, and save payload bytes to the provisional payload (overloaded with input state in C), as indicated by the argument of the *save* command. When the header (regular expression `ueTL`) has been recognized, check whether the payload length L is zero. If so, collect two bytes for a checksum.; if not, and if L is not one, collect payload fields and then a checksum. For each field, first collect a field-length byte F , then collect $F - 1$ field bytes (vertices E and F). When $\Sigma == F$ and $\Sigma\Sigma \leq L$, the byte under consideration is the next field length F . If $\Sigma\Sigma == L + 1$, the byte under consideration is the first checksum byte. After accumulating L bytes, verify the checksum and go to the terminal *Exit* or *Fault* vertex, washing back $L + 3$ bytes if the first checksum byte fails (vertices F and G) or $L + 4$ if the first checksum bytes succeeds and the second checksum byte fails (vertex H).

TODO: save payload fields in *matches*.

A note on diagrams: out-edges denote the character received on entry to the source vertex. The FSM transitions along one of the out edges, depending on that character received and on other guard logic. For instance, vertex F has four out-edges. On two out-edges, the character received on entry is identified as a new field-length F . On one out-edge, the character received on entry is identified as the first checksum byte. On the last out-edge, the character received on entry is identified as a payload byte for the current field.



Legend	
Regular	Save input
Dashed	Discard input
Thick	Clear all saved
Dotted	No changes
DotDash	Washback

To test this against various faults in the field portion, consider the only three possible cases:

1. there is not enough field data
2. there is just the right amount of field data
3. there is not enough field data

■ Not Enough Field Data

char	vertex	buffers	Σ	$\Sigma\Sigma$	$\Sigma < F$ && $\Sigma\Sigma \leq L$	$\Sigma == F$ && $\Sigma\Sigma \leq L$	$\Sigma\Sigma == L + 1$	ΣF	$\Sigma F > L \parallel$ $\Sigma\Sigma > L$	L	F	PL	B ₁	B ₂
L : 3	D	ueT32pB ₁ B ₂	0							3				
F : 4	E	ueT32pB ₁ B ₂	0	1				2	false	3	2			
PL : p	F	ueT32pB ₁ B ₂	1	2	true	false	false	2		3	2	p		

F : B ₁	F	ueT32pB ₁ B ₂	2	3	false	true	false	2		3	2		
x	E	ueT32pB ₁ B ₂	0	4				2 + B ₁	true	3	B ₁		

■ Just Enough Field Data

char	vertex	buffers	Σ	$\Sigma\Sigma$	$\Sigma < F$ && $\Sigma\Sigma \leq L$	$\Sigma == F$ && $\Sigma\Sigma \leq L$	$\Sigma\Sigma == L + 1$	ΣF	$\Sigma F > L \mid \mid$ $\Sigma\Sigma > L$	L	F	PL	B ₁	B ₂
L : 3	D	ueT33ppB ₁ B ₂		0						3				
F : 3	E	ueT33ppB ₁ B ₂	0	1				3	false	3	3			
PL : p	F	ueT33ppB ₁ B ₂	1	2	true	false	false	3		3	3	p		
PL : p	F	ueT33ppB ₁ B ₂	2	3	true	false	false	3		3	3	p		
PL : p	F	ueT33ppB ₁ B ₂	3	4	false	false	true	3		3	3		B ₁	
B ₂ : B ₂	H	ueT33ppB ₁ B ₂	3	4										B ₁

■ Too Much Field Data

□ Without Checking ΣF

This shows the necessity of tracking the sum of the field-length bytes and of checking the invariant

char	vertex	buffers	Σ	$\Sigma\Sigma$	$\Sigma < F$ && $\Sigma\Sigma \leq L$	$\Sigma == F$ && $\Sigma\Sigma \leq L$	$\Sigma\Sigma == L + 1$	ΣF	L	F	PL	B ₁	B ₂
L : 3	D	ueT34pppB ₁ B ₂		0					3				
F : 4	E	ueT34pppB ₁ B ₂	0	1				4	3	4			
PL : p	F	ueT34pppB ₁ B ₂	1	2	true	false	false	4	3	4	p		
PL : p	F	ueT34pppB ₁ B ₂	2	3	true	false	false	4	3	4	p		
PL : p	F	ueT34pppB ₁ B ₂	3	4	false	false	true	4	3	4		p	
B ₂ : B ₁	H	ueT34pppB ₁ B ₂	3	4									B ₁

□ With Checking ΣF

char	vertex	buffers	Σ	$\Sigma\Sigma$	ΣF	$\Sigma F > L$	L	F	PL	B ₁	B ₂
L : 3	D	ueT34pppB ₁ B ₂		0			3				
F : 4	E	ueT34pppB ₁ B ₂	0	1	4	true	3	4			

In[386]:=

```
ClearAll[fletcher];
Module[{l, r},
  fletcher[{li_, ri_}, c_] :=
    (l = Mod[(li + c), 256];
     r = Mod[(ri + l), 256];
     {l, r});
```

In[388]:=

```
ClearAll[newFsmStep];
With[{
  uCode = (*117*)First@ToCharacterCode["u"],
```

```

eCode = (*101*)First@ToCharacterCode["e"]},
(* Static locals. *)
Module[{Σ, ΣΣ, ΣF, T, L, F, B1, B2, C1, C2},

newFsmStep[
  {oldVertex_,
   ignoredActionParam_,
   ignoredActionParamParam_,
   matchesParam_},
  character_] :=

(* non-static locals. *)
Module[{
  matches = matchesParam,
  newVertex,
  action,
  actionArgument},

{newVertex, action, actionArgument} =

Switch[oldVertex,

  "Entry",
  (Σ = 0;
   ΣΣ = 0;
   ΣF = 0;
   T = 0;
   L = 0;
   F = 0;
   B1 = 0;
   B2 = 0;
   C1 = 0;
   C2 = 0;
   {"A", "no action", "no arg"}),

  "A",
  (Switch[character,
    uCode,
    ({C1, C2} = fletcher[{C1, C2}, character];
     {"B", "save", "validated"}),

    default_,
    ({"A", "discard", "no arg"}))],

```

```

"B",
(Switch[character,
  uCode,
  ({"B", "discard", "no arg"}),

  eCode,
  ({C1, C2} = fletcher[{C1, C2}, character];
  {"C", "save", "validated"}),

  default_,
  ({C1, C2} = {0, 0};
  {"A", "clear", "no arg"}))],

"C",
(matches["T"] = T = character;
{C1, C2} = fletcher[{C1, C2}, character];
{"D", "save", "validated"}),

"D",
( $\Sigma$  = 0; matches["L"] = L = character;
{C1, C2} = fletcher[{C1, C2}, character];
If[L == 1,
  {"Fault", "clear", "no arg"},
If[L > 1,
  {"E", "save", "validated"},
If[L == 0,
  {"G", "save", "validated"},
  (*Don't need the following in C language,
  because L is unsigned int.*)
  Throw["@D: Illegal value of L: "<>ToString[L]]]]),

"E",
(matches["F"] = F = character;
 $\Sigma$  = 0;
 $\Sigma\Sigma$ ++;
 $\Sigma F$  += F;
{C1, C2} = fletcher[{C1, C2}, character];
If[F == 0 ||  $\Sigma\Sigma$  > L ||  $\Sigma F$  > L,
(matches["fault"] =
  "@E: F==0? "<>ToString[F == 0] <>
  ";  $\Sigma\Sigma$ >L? "<>ToString[ $\Sigma\Sigma$  > L] <>
  ";  $\Sigma F$ >L? "<>ToString[ $\Sigma F$  > L] <>
  "; washing back: "<>ToString[ $\Sigma\Sigma$  + 2];
  {"Fault", "washback",  $\Sigma\Sigma$  + 2})),

```

```

    ({"F", "save", "provisional"}))],

"F",
( $\Sigma$ ++;
 $\Sigma\Sigma$ ++;
(*character identified as first checksum byte*)
If[ $\Sigma\Sigma$  == L + 1,
  (matches["B1"] = B1 = character;
  If[B1 == C1,
    {"H", "save", "provisional"},
    (matches["fault"] = "@F: invalid checksum: C1 = "<>
      ToString[C1]<>" ; B1 = "<> ToString[B1];
    {"Fault", "washback", L + 3}]]),
  ((*character provisionally identified as new F*)
  If[ $\Sigma$  == F &&  $\Sigma\Sigma$  ≤ L,
    (matches["F"] = F = character;
    {C1, C2} = fletcher[{C1, C2}, character];
    {"E", "save", "provisional"}),
    ((*character identified as payload byte*)
    If[ $\Sigma$  < F &&  $\Sigma\Sigma$  ≤ L,
      (matches["payload"] = character;
      {C1, C2} = fletcher[{C1, C2}, character];
      {"F", "save", "provisional"}),
      Throw["@F: Illegal value of F = "<>
        ToString[L]<>" or  $\Sigma$  = "<> ToString[ $\Sigma$ ]]]]))],

"G",
(matches["B1"] = B1 = character;
If[B1 == C1,
  ({"H", "save", "provisional"}),
  {"Fault", "washback", 3}]]),

"H",
(matches["B2"] = B2 = character;
If[B2 == C2,
  {"Exit", "save", "provisional"}),
(matches["fault"] = "@H: invalid checksum: C2 = "<>
  ToString[C2]<>" ; B2 = "<> ToString[B2];
  {"Fault", "washback", L + 4}]]),

"Fault",
({"Fault", "no action", "no arg"}),

"Exit",

```

```

    ({"Exit", "no action", "no arg"}),

    default_,
    (Throw["Illegal fsm vertex: "<>ToString@oldVertex])
  ];

  Print[<|
    "current vertex" → oldVertex,
    "character" → character,
    "action" → action,
    "action argument" → actionArgument,
    "new vertex" → newVertex,
    "old matches" → matchesParam,
    "B1" → B1, "B2" → B2,
    "C1" → C1, "C2" → C2,
    "Σ" → Σ, "ΣΣ" → ΣΣ, "ΣF" → ΣF,
    "new matches" → matches|>];

    (*return*)
    {newVertex, action, actionArgument, matches}
  ]];

```

Testing the New FSM

■ Old Cases

□ A Single, Invalid Input Character

In[390]:=

```

expect[testScanner["", "j", newFsmStep],
  {<|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {106, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 1, "tail" → 1|>, <|"discarded" → 1, "saved" → 0|>, <||>}]

```

```
<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>
```

```
<|current vertex → A, character → 106, action → discard,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>
```

Out[390]=

PASS

□ A Single, Valid Input Character

In[391]:=

```
expect[testScanner["", "u", newFsmStep],
  {<|"cap" → 16, "origin" → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 1|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 1, "tail" → 1|>, <|"discarded" → 0, "saved" → 1|>, <| |>}]
```

```
<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>
```

```
<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>
```

Out[391]=

PASS

□ A Complete, Valid Sequence

In[392]:=

```
expect[testScanner["", "ueJK", newFsmStep],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 4, "tail" → 4|>, <|"discarded" → 0, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 74, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 36, C2 → 115, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74|>|>

<|current vertex → D, character → 75, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
  B2 → 0, C1 → 111, C2 → 226, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74, L → 75|>|>

```

Out[392]=

PASS

□ Incomplete Scanner State

In[393]:=

```

expect[testScanner["u", "", newFsmStep],
  {<|"cap" → 16, "origin" → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 1|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"discarded" → 0, "saved" → 0|>, <| |>}]

```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

```

Out[393]=

PASS

□ Stuttering

In[394]:=

```
expect[testScanner["u", "uuuuuuueJK", newFsmStep],  
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},  
    "head" → 0, "tail" → 4|>, <|"cap" → 16,  
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,  
    <|"cap" → 16, "origin" → {117, 117, 117, 117, 117, 117, 117,  
      101, 74, 75, 0, 0, 0, 0, 0, 0}, "head" → 10, "tail" → 10|>,  
    <|"discarded" → 7, "saved" → 3|>, <|"T" → 74, "L" → 75|>}]
```



```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 74, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 36, C2 → 115, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74|>|>

<|current vertex → D, character → 75, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
  B2 → 0, C1 → 111, C2 → 226, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74, L → 75|>|>

```

Out[394]=

PASS

□ Scanner State with Payload Characters

The results differ from the first FSM because the new FSM interprets the additional characters as payload.

In[395]:=

```
testScanner["ue", "xyzabcdef", newFsmStep]
```

```
<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 120, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 82, C2 → 161, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 120|>|>

<|current vertex → D, character → 121, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 120|>, B1 → 0,
  B2 → 0, C1 → 203, C2 → 108, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 120, L → 121|>|>

<|current vertex → E, character → 122, action → washback, action argument → 3,
  new vertex → Fault, old matches → <|T → 120, L → 121|>, B1 → 0, B2 → 0, C1 → 69,
  C2 → 177, Σ → 0, ΣΣ → 1, ΣF → 122, new matches → <|T → 120, L → 121, F → 122,
  fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>|>
```

Out[395]=

```
{<|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 0|>, <|cap → 16,
  origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
  <|cap → 16, origin → {120, 121, 122, 97, 98, 99, 100, 101, 102, 0, 0, 0, 0, 0, 0, 0},
  head → 3, tail → 9|>, <|discarded → 0, saved → 2|>, <|T → 120, L → 121, F → 122,
  fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>|>}
```

In[396]:=

```
expect[testScanner["ue", "xyzabcdef", newFsmStep],
  {<|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {120, 121, 122, 97, 98, 99, 100,
      101, 102, 0, 0, 0, 0, 0, 0, 0}, "head" → 3, "tail" → 9|>,
    <|"discarded" → 0, "saved" → 2|>, <|"T" → 120, "L" → 121, "F" → 122,
    "fault" → "@E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3"|>|>}}
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 120, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 82, C2 → 161, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 120|>|>

<|current vertex → D, character → 121, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 120|>, B1 → 0,
  B2 → 0, C1 → 203, C2 → 108, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 120, L → 121|>|>

<|current vertex → E, character → 122, action → washback, action argument → 3,
  new vertex → Fault, old matches → <|T → 120, L → 121|>, B1 → 0, B2 → 0, C1 → 69,
  C2 → 177, Σ → 0, ΣΣ → 1, ΣF → 122, new matches → <|T → 120, L → 121, F → 122,
  fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>|>

```

Out[396]=

PASS

In[397]:=

```

ClearAll[dpyScanner];
dpyScanner[scannerChars_String, inputChars_String, fsmStepper_ : fsmStep] :=
Module[{ssPrior, isPrior,
  ss = initializedCB[scannerChars],
  ps = initializedCB[""],
  is = initializedCB[inputChars],
  metrics,
  matches},
ssPrior = ss;
isPrior = is;
{ss, ps, is, metrics, matches} = scanner[ss, ps, is, fsmStepper];
Column[{
  "***** BEFORE: initial scanner state *****",
  CBdisplay[ssPrior],
  "***** BEFORE: initial input state *****",
  CBdisplay[isPrior],
  "***** AFTER: scanner state *****",
  CBdisplay[ss],
  "***** AFTER: provisional payload *****",
  CBdisplay[ps],
  "***** AFTER: input state *****",
  CBdisplay[is],
  "***** metrics *****",
  metrics,
  "***** matches *****",
  matches}]];

```

The next two samples pick up a bad F and throw it away after invalidating the prefixes.

In[399]:=

```
dpyScanner["ue", "xyzabcdef", newFsmStep]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 120, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 82, C2 → 161, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 120|>|>

<|current vertex → D, character → 121, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 120|>, B1 → 0,
  B2 → 0, C1 → 203, C2 → 108, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 120, L → 121|>|>

<|current vertex → E, character → 122, action → washback, action argument → 3,
  new vertex → Fault, old matches → <|T → 120, L → 121|>, B1 → 0, B2 → 0, C1 → 69,
  C2 → 177, Σ → 0, ΣΣ → 1, ΣF → 122, new matches → <|T → 120, L → 121, F → 122,
  fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>|>

```

Out[399]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:
***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 120 121 122 97 98 99 100 101 102 0  0  0  0  0  0  0
head:   ^
tail:
***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 120 121 122 97 98 99 100 101 102 0  0  0  0  0  0  0
head:
tail:
***** metrics *****
<|discarded → 0, saved → 2|>
***** matches *****
<|T → 120, L → 121, F → 122,
  fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>

```

In[400]:=

```
dpyScanner["ueJ", "KZABCDEF", newFsmStep]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 74, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 36, C2 → 115, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74|>|>

<|current vertex → D, character → 75, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
  B2 → 0, C1 → 111, C2 → 226, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74, L → 75|>|>

<|current vertex → E, character → 90, action → washback, action argument → 3,
  new vertex → Fault, old matches → <|T → 74, L → 75|>, B1 → 0, B2 → 0, C1 → 201,
  C2 → 171, Σ → 0, ΣΣ → 1, ΣF → 90, new matches → <|T → 74, L → 75, F → 90,
  fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>|>

```

Out[400]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 74 0 0 0 0 0 0 0 0 0 0 0 0
head:   ^
tail:   ^
***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 75 90 65 66 67 68 69 70 0 0 0 0 0 0 0
head:   ^
tail:   ^
***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
head:   ^
tail:   ^
***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
head:   ^
tail:   ^
***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 75 90 65 66 67 68 69 70 0 0 0 0 0 0 0
head:   ^
tail:   ^
***** metrics *****
<|discarded → 0, saved → 1|>
***** matches *****
<|T → 74, L → 75, F → 90,
  fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>

```

In[401]:=

```

expect[testScanner["ueJ", "KZABCDEF", newFsmStep],
  {<|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {75, 90, 65, 66, 67, 68, 69, 70, 0, 0, 0, 0, 0, 0, 0},
    "head" → 2, "tail" → 8|>,
    <|"discarded" → 0, "saved" → 1|>, <|"T" → 74, "L" → 75, "F" → 90,
    "fault" → "@E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3"|>}}]

```



```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 74, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 36, C2 → 115, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74|>|>

<|current vertex → D, character → 75, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
  B2 → 0, C1 → 111, C2 → 226, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74, L → 75|>|>

<|current vertex → E, character → 90, action → washback, action argument → 3,
  new vertex → Fault, old matches → <|T → 74, L → 75|>, B1 → 0, B2 → 0, C1 → 201,
  C2 → 171, Σ → 0, ΣΣ → 1, ΣF → 90, new matches → <|T → 74, L → 75, F → 90,
  fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>|>

```

Out[401]=

PASS

□ A Little Junk in the Input

In[402]:=

```
dpyScanner["u", "asdfueJK", newFsmStep]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 97, action → clear,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 115, action → discard,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 100, action → discard,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 102, action → discard,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 74, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 36, C2 → 115, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74|>|>

<|current vertex → D, character → 75, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
  B2 → 0, C1 → 111, C2 → 226, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74, L → 75|>|>

```

Out[402]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 97 115 100 102 117 101 74 75 0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 74 75 0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 97 115 100 102 117 101 74 75 0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** metrics *****
<|discarded → 4, saved → 4|>

***** matches *****
<|T → 74, L → 75|>

```

In[403]=

```

expect[testScanner["u", "asdfueJK", newFsmStep],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {97, 115, 100, 102, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 8, "tail" → 8|>, <|"discarded" → 4, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]

```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 97, action → clear,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 115, action → discard,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 100, action → discard,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 102, action → discard,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 74, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 36, C2 → 115, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74|>|>

<|current vertex → D, character → 75, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
  B2 → 0, C1 → 111, C2 → 226, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74, L → 75|>|>

```

Out[403]=

PASS

In[404]:=

dpyScanner["u", "uXXuueJK", newFsmStep]

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 88, action → clear,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 88, action → discard,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 74, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 36, C2 → 115, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74|>|>

<|current vertex → D, character → 75, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
  B2 → 0, C1 → 111, C2 → 226, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74, L → 75|>|>

```

Out[404]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 88 88 117 117 101 74 75 0 0 0 0 0 0 0 0
head:   ^
tail:   ^

***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 74 75 0 0 0 0 0 0 0 0 0 0 0 0
head:   ^
tail:   ^

***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 88 88 117 117 101 74 75 0 0 0 0 0 0 0 0
head:   ^
tail:   ^

***** metrics *****
<|discarded → 4, saved → 4|>

***** matches *****
<|T → 74, L → 75|>

```

In[405]=

```

expect[testScanner["u", "uXXuueJK", newFsmStep],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 88, 88, 117, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 8, "tail" → 8|>, <|"discarded" → 4, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]

```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 88, action → clear,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 88, action → discard,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 117, action → discard,
  action argument → no arg, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 74, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 36, C2 → 115, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74|>|>

<|current vertex → D, character → 75, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
  B2 → 0, C1 → 111, C2 → 226, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74, L → 75|>|>

```

Out[405]=

PASS

□ Lots of Junk in the Input

In[406]:=

```
dpyScanner["u", "uuuuuuuXXxueJK", newFsmStep]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

```


<|current vertex → D, character → 75, action → save,
 action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
 B2 → 0, C1 → 111, C2 → 226, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 74, L → 75|>|>

Out[406]=

```
***** BEFORE: initial scanner state *****
```

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
origin:	117	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
head:	^															
tail:		^														

```
***** BEFORE: initial input state *****
```

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
origin:	117	117	117	117	117	117	117	88	88	88	117	117	101	74	75	0
head:	^															
tail:																^

```
***** AFTER: scanner state *****
```

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
origin:	117	101	74	75	0	0	0	0	0	0	0	0	0	0	0	0
head:	^															
tail:					^											

```
***** AFTER: provisional payload *****
```

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
origin:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
head:	^															
tail:	^															

```
***** AFTER: input state *****
```

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
origin:	117	117	117	117	117	117	117	88	88	88	117	117	101	74	75	0
head:																^
tail:																^

```
***** metrics *****
<|discarded → 11, saved → 4|>
***** matches *****
<|T → 74, L → 75|>
```

In[407]:=

```
expect[testScanner["u", "uuuuuuuXXXuueJK", newFsmStep],
  {<|"cap" → 16, "origin" → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {117, 117, 117, 117, 117, 117, 117, 88,
      88, 88, 117, 117, 101, 74, 75, 0}, "head" → 15, "tail" → 15|>,
    <|"discarded" → 11, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

<|current vertex → Entry, character → Null, action → no action,
 action argument → no arg, new vertex → A, old matches → <|>, B1 → 0,
 B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|>|>


```

<|current vertex → C, character → 74, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 36, C2 → 115,  $\Sigma$  → 0,  $\Sigma\Sigma$  → 0,  $\Sigma F$  → 0, new matches → <|T → 74|>|>
<|current vertex → D, character → 75, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 74|>, B1 → 0,
  B2 → 0, C1 → 111, C2 → 226,  $\Sigma$  → 0,  $\Sigma\Sigma$  → 0,  $\Sigma F$  → 0, new matches → <|T → 74, L → 75|>|>

```

Out[407]=

PASS

□ All Bets Are Off

Input-state circular buffer is full, therefore it is also empty. Caller has violated the contract by not ensuring valid contents, so caller should not expect consistent results

In[408]:=

```
dpyScanner["u", "uuuuuuuXXXXuueJK", newFsmStep]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

```

Out[408]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 117 117 117 117 117 117 88 88 88 88 117 117 101 74 75
head:   ^
tail:   ^

***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 117 117 117 117 117 117 88 88 88 88 117 117 101 74 75
head:   ^
tail:   ^

***** metrics *****
<|discarded → 0, saved → 0|>

***** matches *****
<| |>

```

In[409]:=

```

expect[testScanner["u", "uuuuuuuXXXXuueJK", newFsmStep],
  {<|"cap" → 16, "origin" → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 1|>,
  <|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"cap" → 16, "origin" →
      {117, 117, 117, 117, 117, 117, 117, 88, 88, 88, 88, 117, 117, 101, 74, 75},
    "head" → 0, "tail" → 0|>, <|"discarded" → 0, "saved" → 0|>, <| |>}]

```



```

<|cap → 16, origin → {120, 121, 122, 97, 98, 99, 100, 101, 102, 0, 0, 0, 0, 0, 0},
  head → 2, tail → 9|>, <|discarded → 0, saved → 2|>, <|T → 120, L → 121|>},
{<|cap → 16, origin → {117, 101, 88, 89, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 4|>, <|cap → 16,
  origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
<|cap → 16, origin → {89, 90, 65, 66, 67, 68, 69, 70, 0, 0, 0, 0, 0, 0, 0},
  head → 1, tail → 8|>, <|discarded → 0, saved → 1|>, <|T → 88, L → 89|>},
{<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 4|>, <|cap → 16,
  origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
<|cap → 16, origin → {97, 115, 100, 102, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0},
  head → 8, tail → 8|>, <|discarded → 4, saved → 4|>, <|T → 74, L → 75|>},
{<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 4|>, <|cap → 16,
  origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
<|cap → 16, origin → {88, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 5, tail → 5|>, <|discarded → 1, saved → 4|>, <|T → 74, L → 75|>},
{<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 4|>, <|cap → 16,
  origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
<|cap → 16, origin → {117, 88, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 6, tail → 6|>, <|discarded → 2, saved → 4|>, <|T → 74, L → 75|>},
{<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 4|>, <|cap → 16,
  origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
<|cap → 16, origin → {117, 88, 88, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0},
  head → 7, tail → 7|>, <|discarded → 3, saved → 4|>, <|T → 74, L → 75|>},
{<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 4|>, <|cap → 16,
  origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
<|cap → 16, origin → {117, 88, 88, 117, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0},
  head → 8, tail → 8|>, <|discarded → 4, saved → 4|>, <|T → 74, L → 75|>},
{<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0,
  tail → 4|>, <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 0|>, <|cap → 16, origin →
  {117, 117, 117, 117, 117, 117, 117, 117, 88, 88, 88, 117, 101, 74, 75, 0},
  head → 15, tail → 15|>, <|discarded → 11, saved → 4|>, <|T → 74, L → 75|>},
{<|cap → 16, origin → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0,
  tail → 1|>, <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 0|>, <|cap → 16, origin →
  {117, 117, 117, 117, 117, 117, 117, 117, 88, 88, 88, 88, 117, 117, 101, 74, 75},
  head → 0, tail → 0|>, <|discarded → 0, saved → 0|>, <|>},
{<|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  head → 0, tail → 0|>, <|cap → 16,

```

```

    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {106, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 1, tail → 1|>, <|discarded → 1, saved → 0|>, <| |>,
    {<|cap → 16, origin → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 1|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 1, tail → 1|>, <|discarded → 0, saved → 1|>, <| |>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 4, tail → 4|>, <|discarded → 0, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 1|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|discarded → 0, saved → 0|>, <| |>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0,
    tail → 4|>, <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|cap → 16,
    origin → {117, 117, 117, 117, 117, 117, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0},
    head → 10, tail → 10|>, <|discarded → 7, saved → 3|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {120, 121, 122, 97, 98, 99, 100, 101, 102, 0, 0, 0, 0, 0, 0},
    head → 3, tail → 9|>, <|discarded → 0, saved → 2|>, <|T → 120, L → 121, F → 122,
    fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>},
    {<|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {75, 90, 65, 66, 67, 68, 69, 70, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 2, tail → 8|>, <|discarded → 0, saved → 1|>, <|T → 74, L → 75, F → 90,
    fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 3|>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {97, 115, 100, 102, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 8, tail → 8|>, <|discarded → 4, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 4|>, <|cap → 16,
    origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 88, 88, 117, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0},

```

```

    head → 8, tail → 8|>, <|discarded → 4, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0,
    tail → 4|>, <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|cap → 16, origin →
    {117, 117, 117, 117, 117, 117, 117, 88, 88, 88, 117, 117, 101, 74, 75, 0},
    head → 15, tail → 15|>, <|discarded → 11, saved → 4|>, <|T → 74, L → 75|>},
    {<|cap → 16, origin → {117, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0,
    tail → 1|>, <|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 0, tail → 0|>, <|cap → 16, origin →
    {117, 117, 117, 117, 117, 117, 117, 88, 88, 88, 88, 117, 117, 101, 74, 75},
    head → 0, tail → 0|>, <|discarded → 0, saved → 0|>, <|>}}, failures → {}|>

```

■ New Cases

To test the bottom of the new FSM, the part that saves fields, we need new support routines that take numerical characters without converting them to character codes.

In[411]:=

```

ClearAll[initializedCBWithNumbers];
initializedCBWithNumbers[ns_List] :=
  Fold[CBpush[#1, #2] &, makeTestCB[16], ns];

```

In[413]:=

```

ClearAll[testWithNumbers];
testWithNumbers[scannerNumbers_List, inputNumbers_List, fsmStepper_] :=
  Module[{
    ss = initializedCBWithNumbers[scannerNumbers],
    ps = initializedCBWithNumbers[{}],
    is = initializedCBWithNumbers[inputNumbers],
    metrics,
    matches},
    {ss, ps, is, metrics, matches} = scanner[ss, ps, is, fsmStepper]];

```


In[415]:=

```

ClearAll[dpyScannerWithNumbers];
dpyScannerWithNumbers[scannerNumbers_List, inputNumbers_List, fsmStepper_] :=
Module[{ssPrior, isPrior,
  ss = initializedCBWithNumbers[scannerNumbers],
  ps = initializedCBWithNumbers[{}],
  is = initializedCBWithNumbers[inputNumbers],
  metrics,
  matches},
ssPrior = ss;
isPrior = is;
{ss, ps, is, metrics, matches} = scanner[ss, ps, is, fsmStepper];
Column[{
  "***** BEFORE: initial scanner state *****",
  CBdisplay[ssPrior],
  "***** BEFORE: initial input state *****",
  CBdisplay[isPrior],
  "***** AFTER: scanner state *****",
  CBdisplay[ss],
  "***** AFTER: provisional payload *****",
  CBdisplay[ps],
  "***** AFTER: input state *****",
  CBdisplay[is],
  "***** metrics *****",
  metrics,
  "***** matches *****",
  matches}]];

```

In[417]:=

```

dpyScannerWithNumbers[{117, 101, 0, 255}, {}, newFsmStep]

```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 0, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 41, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 0|>|>

<|current vertex → D, character → 255, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 0|>, B1 → 0,
  B2 → 0, C1 → 217, C2 → 2, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 0, L → 255|>|>

```

Out[417]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 0 255 0 0 0 0 0 0 0 0 0 0 0 0
head:   ^
tail:   ^

***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 0 255 0 0 0 0 0 0 0 0 0 0 0 0
head:   ^
tail:   ^

***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^

***** metrics *****
<|discarded → 0, saved → 0|>

***** matches *****
<|T → 0, L → 255|>

```

In[418]:=

```
expect[testWithNumbers[{117, 101, 0, 255}, {}], newFsmStep],
  {<|"cap" → 16, "origin" → {117, 101, 0, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 4|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"discarded" → 0, "saved" → 0|>, <|"T" → 0, "L" → 255|>}]
```

```
<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 0, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 41, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 0|>|>

<|current vertex → D, character → 255, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 0|>, B1 → 0,
  B2 → 0, C1 → 217, C2 → 2, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 0, L → 255|>|>
```

Out[418]=

PASS

□ Correctly parses a valid packet

In[419]:=

```
dpyScannerWithNumbers[{117, 101, 1, 2, 2, 1, 16^^E0, 16^^C6}, {}], newFsmStep]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 1, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 219, C2 → 42, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1|>|>

<|current vertex → D, character → 2, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 1|>, B1 → 0,
  B2 → 0, C1 → 221, C2 → 7, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1, L → 2|>|>

<|current vertex → E, character → 2, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2|>, B1 → 0, B2 → 0, C1 → 223,
  C2 → 230, Σ → 0, ΣΣ → 1, ΣF → 2, new matches → <|T → 1, L → 2, F → 2|>|>

<|current vertex → F, character → 1, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2, F → 2|>, B1 → 0, B2 → 0, C1 → 224,
  C2 → 198, Σ → 1, ΣΣ → 2, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1|>|>

<|current vertex → F, character → 224, action → save, action argument → provisional,
  new vertex → H, old matches → <|T → 1, L → 2, F → 2, payload → 1|>, B1 → 224, B2 → 0, C1 → 224,
  C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>|>

<|current vertex → H, character → 198, action → save, action argument → provisional,
  new vertex → Exit, old matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>,
  B1 → 224, B2 → 198, C1 → 224, C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2,
  new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>|>

```

Out[419]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1 224 198 0  0  0  0  0  0  0  0
head:   ^
tail:
***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1 224 198 0  0  0  0  0  0  0  0
head:   ^
tail:
***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** metrics *****
<|discarded → 0, saved → 0|>
***** matches *****
<|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>

```

In[420]:=

```

expect[testWithNumbers[{117, 101, 1, 2, 2, 1, 16^E0, 16^C6}, {}, newFsmStep],
  {<|"cap" → 16, "origin" → {117, 101, 1, 2, 2, 1, 224, 198, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 8|>, <|"cap" → 16,
    "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, "head" → 0, "tail" → 0|>,
    <|"cap" → 16, "origin" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"discarded" → 0, "saved" → 0|>,
    <|"T" → 1, "L" → 2, "F" → 2, "payload" → 1, "B1" → 224, "B2" → 198|>}]

```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 1, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 219, C2 → 42, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1|>|>

<|current vertex → D, character → 2, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 1|>, B1 → 0,
  B2 → 0, C1 → 221, C2 → 7, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1, L → 2|>|>

<|current vertex → E, character → 2, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2|>, B1 → 0, B2 → 0, C1 → 223,
  C2 → 230, Σ → 0, ΣΣ → 1, ΣF → 2, new matches → <|T → 1, L → 2, F → 2|>|>

<|current vertex → F, character → 1, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2, F → 2|>, B1 → 0, B2 → 0, C1 → 224,
  C2 → 198, Σ → 1, ΣΣ → 2, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1|>|>

<|current vertex → F, character → 224, action → save, action argument → provisional,
  new vertex → H, old matches → <|T → 1, L → 2, F → 2, payload → 1|>, B1 → 224, B2 → 0, C1 → 224,
  C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>|>

<|current vertex → H, character → 198, action → save, action argument → provisional,
  new vertex → Exit, old matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>,
  B1 → 224, B2 → 198, C1 → 224, C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2,
  new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>|>

```

Out[420]=

PASS

□ With Bookend Partitioning

In[421]:=

```
dpyScannerWithNumbers[{117, 101, 1, 2, 2, 1, 16^E0}, {16^C6}, newFsmStep]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 1, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 219, C2 → 42, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1|>|>

<|current vertex → D, character → 2, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 1|>, B1 → 0,
  B2 → 0, C1 → 221, C2 → 7, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1, L → 2|>|>

<|current vertex → E, character → 2, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2|>, B1 → 0, B2 → 0, C1 → 223,
  C2 → 230, Σ → 0, ΣΣ → 1, ΣF → 2, new matches → <|T → 1, L → 2, F → 2|>|>

<|current vertex → F, character → 1, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2, F → 2|>, B1 → 0, B2 → 0, C1 → 224,
  C2 → 198, Σ → 1, ΣΣ → 2, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1|>|>

<|current vertex → F, character → 224, action → save, action argument → provisional,
  new vertex → H, old matches → <|T → 1, L → 2, F → 2, payload → 1|>, B1 → 224, B2 → 0, C1 → 224,
  C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>|>

<|current vertex → H, character → 198, action → save, action argument → provisional,
  new vertex → Exit, old matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>,
  B1 → 224, B2 → 198, C1 → 224, C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2,
  new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>|>

```

Out[421]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1 224 0  0  0  0  0  0  0  0
head:   ^
tail:
***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 198 0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1 224 198 0  0  0  0  0  0  0
head:   ^
tail:
***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 198 0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:
tail:   ^
***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 198 0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:
tail:   ^
***** metrics *****
<|discarded → 0, saved → 0|>
***** matches *****
<|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>

```

In[422]=

```

dpyScannerWithNumbers[{117, 101, 1, 2, 2, 1}, {16^E0, 16^C6}, newFsmStep]

```



```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 1, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 219, C2 → 42, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1|>|>

<|current vertex → D, character → 2, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 1|>, B1 → 0,
  B2 → 0, C1 → 221, C2 → 7, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1, L → 2|>|>

<|current vertex → E, character → 2, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2|>, B1 → 0, B2 → 0, C1 → 223,
  C2 → 230, Σ → 0, ΣΣ → 1, ΣF → 2, new matches → <|T → 1, L → 2, F → 2|>|>

<|current vertex → F, character → 1, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2, F → 2|>, B1 → 0, B2 → 0, C1 → 224,
  C2 → 198, Σ → 1, ΣΣ → 2, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1|>|>

<|current vertex → F, character → 224, action → save, action argument → provisional,
  new vertex → H, old matches → <|T → 1, L → 2, F → 2, payload → 1|>, B1 → 224, B2 → 0, C1 → 224,
  C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>|>

<|current vertex → H, character → 198, action → save, action argument → provisional,
  new vertex → Exit, old matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>,
  B1 → 224, B2 → 198, C1 → 224, C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2,
  new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>|>

```

Out[422]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:
***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 224 198 0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1 224 198 0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 224 198 0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 224 198 0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:   ^
***** metrics *****
<|discarded → 0, saved → 0|>
***** matches *****
<|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>

```

In[423]:=

```

dpyScannerWithNumbers[{117, 101, 1, 2, 2}, {1, 16^E0, 16^C6}, newFsmStep]

```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 1, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 219, C2 → 42, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1|>|>

<|current vertex → D, character → 2, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 1|>, B1 → 0,
  B2 → 0, C1 → 221, C2 → 7, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1, L → 2|>|>

<|current vertex → E, character → 2, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2|>, B1 → 0, B2 → 0, C1 → 223,
  C2 → 230, Σ → 0, ΣΣ → 1, ΣF → 2, new matches → <|T → 1, L → 2, F → 2|>|>

<|current vertex → F, character → 1, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2, F → 2|>, B1 → 0, B2 → 0, C1 → 224,
  C2 → 198, Σ → 1, ΣΣ → 2, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1|>|>

<|current vertex → F, character → 224, action → save, action argument → provisional,
  new vertex → H, old matches → <|T → 1, L → 2, F → 2, payload → 1|>, B1 → 224, B2 → 0, C1 → 224,
  C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>|>

<|current vertex → H, character → 198, action → save, action argument → provisional,
  new vertex → Exit, old matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>,
  B1 → 224, B2 → 198, C1 → 224, C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2,
  new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>|>

```

Out[423]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:
***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 1 224 198 0  0  0  0  0  0  0  0  0  0  0  0  0
head:   ^
tail:
***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1 224 198 0  0  0  0  0  0  0  0
head:   ^
tail:
***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 1 224 198 0  0  0  0  0  0  0  0  0  0  0  0  0
head:
tail:
***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 1 224 198 0  0  0  0  0  0  0  0  0  0  0  0  0
head:
tail:
***** metrics *****
<|discarded → 0, saved → 0|>
***** matches *****
<|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>

```

In[424]:=

```
dpyScannerWithNumbers[{}, {117, 101, 1, 2, 2, 1, 16^^E0, 16^^C6}, newFsmStep]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 1, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 219, C2 → 42, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1|>|>

<|current vertex → D, character → 2, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 1|>, B1 → 0,
  B2 → 0, C1 → 221, C2 → 7, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1, L → 2|>|>

<|current vertex → E, character → 2, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2|>, B1 → 0, B2 → 0, C1 → 223,
  C2 → 230, Σ → 0, ΣΣ → 1, ΣF → 2, new matches → <|T → 1, L → 2, F → 2|>|>

<|current vertex → F, character → 1, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 2, F → 2|>, B1 → 0, B2 → 0, C1 → 224,
  C2 → 198, Σ → 1, ΣΣ → 2, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1|>|>

<|current vertex → F, character → 224, action → save, action argument → provisional,
  new vertex → H, old matches → <|T → 1, L → 2, F → 2, payload → 1|>, B1 → 224, B2 → 0, C1 → 224,
  C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2, new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>|>

<|current vertex → H, character → 198, action → save, action argument → provisional,
  new vertex → Exit, old matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224|>,
  B1 → 224, B2 → 198, C1 → 224, C2 → 198, Σ → 2, ΣΣ → 3, ΣF → 2,
  new matches → <|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>|>

```

Out[424]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:    ^
tail:    ^

***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1 224 198  0  0  0  0  0  0  0  0
head:    ^
tail:    ^

***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1 224 198  0  0  0  0  0  0  0  0
head:    ^
tail:    ^

***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin:  2  1 224 198  0  0  0  0  0  0  0  0  0  0  0  0
head:    ^
tail:    ^

***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  2  2  1 224 198  0  0  0  0  0  0  0  0
head:    ^
tail:    ^

***** metrics *****
<|discarded → 0, saved → 4|>

***** matches *****
<|T → 1, L → 2, F → 2, payload → 1, B1 → 224, B2 → 198|>

```

□ Longer Correct Payloads

In[425]:=

```
Fold[fletcher, {0, 0}, {117, 101, 1, 5, 2, 42, 3, 43, 44}]
```

Out[425]=

```
{102, 167}
```

In[426]:=

```
dpyScannerWithNumbers[{}, {117, 101, 1, 5, 2, 42, 3, 43, 44, 102, 167}, newFsmStep]
```

```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 1, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 219, C2 → 42, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1|>|>

<|current vertex → D, character → 5, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 1|>, B1 → 0,
  B2 → 0, C1 → 224, C2 → 10, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1, L → 5|>|>

<|current vertex → E, character → 2, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 5|>, B1 → 0, B2 → 0, C1 → 226,
  C2 → 236, Σ → 0, ΣΣ → 1, ΣF → 2, new matches → <|T → 1, L → 5, F → 2|>|>

<|current vertex → F, character → 42, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 5, F → 2|>, B1 → 0, B2 → 0, C1 → 12,
  C2 → 248, Σ → 1, ΣΣ → 2, ΣF → 2, new matches → <|T → 1, L → 5, F → 2, payload → 42|>|>

<|current vertex → F, character → 3, action → save, action argument → provisional,
  new vertex → E, old matches → <|T → 1, L → 5, F → 2, payload → 42|>, B1 → 0, B2 → 0,
  C1 → 15, C2 → 7, Σ → 2, ΣΣ → 3, ΣF → 2, new matches → <|T → 1, L → 5, F → 3, payload → 42|>|>

<|current vertex → E, character → 43, action → washback, action argument → 6,
  new vertex → Fault, old matches → <|T → 1, L → 5, F → 3, payload → 42|>, B1 → 0,
  B2 → 0, C1 → 58, C2 → 65, Σ → 0, ΣΣ → 4, ΣF → 45, new matches → <|T → 1, L → 5, F → 43,
  payload → 42, fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 6|>|>

```

Out[426]=

```

***** BEFORE: initial scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:    ^
tail:    ^

***** BEFORE: initial input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  5  2  42 3  43 44 102 167 0  0  0  0  0
head:    ^
tail:    ^

***** AFTER: scanner state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
head:    ^
tail:    ^

***** AFTER: provisional payload *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin:  2 42 3  0  0  0  0  0  0  0  0  0  0  0  0  0
head:    ^
tail:    ^

***** AFTER: input state *****
index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
origin: 117 101 1  5  2  2  42 3  44 102 167 0  0  0  0  0
head:    ^
tail:    ^

***** metrics *****
<|discarded → 0, saved → 4|>
***** matches *****
<|T → 1, L → 5, F → 43, payload → 42,
  fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 6|>

```

In[427]:=

```
dpyScannerWithNumbers[{117, 101, 1, 4, 2, 42, 2, 43, 56, 58}, {}, newFsmStep]
```



```

<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → A, character → 117, action → save,
  action argument → validated, new vertex → B, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 117, C2 → 117, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → B, character → 101, action → save,
  action argument → validated, new vertex → C, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 218, C2 → 79, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>

<|current vertex → C, character → 1, action → save,
  action argument → validated, new vertex → D, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 219, C2 → 42, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1|>|>

<|current vertex → D, character → 4, action → save,
  action argument → validated, new vertex → E, old matches → <|T → 1|>, B1 → 0,
  B2 → 0, C1 → 223, C2 → 9, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <|T → 1, L → 4|>|>



<|current vertex → E, character → 2, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 4|>, B1 → 0, B2 → 0, C1 → 225,
  C2 → 234, Σ → 0, ΣΣ → 1, ΣF → 2, new matches → <|T → 1, L → 4, F → 2|>|>

<|current vertex → F, character → 42, action → save, action argument → provisional,
  new vertex → F, old matches → <|T → 1, L → 4, F → 2|>, B1 → 0, B2 → 0, C1 → 11,
  C2 → 245, Σ → 1, ΣΣ → 2, ΣF → 2, new matches → <|T → 1, L → 4, F → 2, payload → 42|>|>

<|current vertex → F, character → 2, action → save, action argument → provisional,
  new vertex → E, old matches → <|T → 1, L → 4, F → 2, payload → 42|>, B1 → 0, B2 → 0,
  C1 → 13, C2 → 2, Σ → 2, ΣΣ → 3, ΣF → 2, new matches → <|T → 1, L → 4, F → 2, payload → 42|>|>

<|current vertex → E, character → 43, action → washback, action argument → 6,
  new vertex → Fault, old matches → <|T → 1, L → 4, F → 2, payload → 42|>, B1 → 0,
  B2 → 0, C1 → 56, C2 → 58, Σ → 0, ΣΣ → 4, ΣF → 45, new matches → <|T → 1, L → 4, F → 43,
  payload → 42, fault → @E: F==0? False; ΣΣ>L? False; ΣF>L? True; washing back: 6|>|>

```

 **Throw** : Uncaught Throw [3: Invalid action washback returned by fsm step on character 43] returned to top level. 

Out[427]=

```
Hold[Throw[3: Invalid action washback returned by fsm step on character 43]]
```

In[428]:=

```
dpyScannerWithNumbers[{{117, 101, 1, 4, 2, 42, 2}}, {43, 56, 58}, newFsmStep]
```

```
<|current vertex → Entry, character → Null, action → no action,
  action argument → no arg, new vertex → A, old matches → <| |>, B1 → 0,
  B2 → 0, C1 → 0, C2 → 0, Σ → 0, ΣΣ → 0, ΣF → 0, new matches → <| |>|>
```

... **Part** : The expression `1 + CBpush [<|cap → 16, origin → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, head → 0, tail → 0|>, {117, 101, 1, 4, 2, 42, 2}][head]` cannot be used as a part specification.

... **\$RecursionLimit** : Recursion depth of 1024 exceeded.



Out[428]=

```
TerminatedEvaluation[RecursionLimit]
```

Inverting the Control

In this design, FSMs are responsible for recognizing characters one at a time and for requesting actions from the calling scanner. The scanner doesn't know much about the FSM: only the list of commands that the FSM may request. The FSM knows nothing about its caller. Thus, the scanner can be tested independently of the FSM.

This could be further generalized by having the FSM return closures to the scanner, rather than strings commands. Such is overkill for the example.

Most parsers work by calling I/O routines such as `getchar` and `pushback`. They take actions directly inline, but are unprepared for i/o faults like exhaustion and overflow. This parser separates those concerns completely.

Sandbox

Stuff I am not ready yet to throw away.

Parser

In[429]:=

```
Manipulate[
  Module[{init, a, b, c, d, e, f, g, h,
    exit, fault, vertices, vertexLabel, edgeLabel, colors},
    vertexLabel = v ↦ Placed[v, Center];
    edgeLabel = {edge, bg} ↦ Framed[edge, Background → bg];
    vertices =
      {init, a, b, c, d, g, e, f, h, exit, fault} = {"Entry", "A", "B", "C", "D\nΣΣ=0",
        "G", "E\nΣ=0\nΣΣ++", "F\nΣ++\nΣΣ++", "H\ncheck-\nsum", "Exit", "Fault"};
    colors = <|"Entry" → LightGray, "A" → LightOrange,
```

```

"B" → LightOrange, "C" → LightOrange, "D\nΣ=0" → LightOrange,
"E\nΣ=0\nΣ++" → LightOrange, "F\nΣ++\nΣ++" → LightOrange, "G" → LightOrange,
"H\ncheck-\nsum" → LightOrange, "Exit" → LightGray, "Fault" → LightGray|>;
Row[{Graph[
  Property[Style[#, colors[#]], VertexLabels → vertexLabel[#]] & /@ vertices,
  {Style[init → a, Dotted],
    Property[Style[a → a, Dashed],
      EdgeLabels → edgeLabel["^u", LightYellow]],
    Property[a → b, EdgeLabels → edgeLabel["u", LightGreen]],
    Property[Style[b → a, Thick], EdgeLabels → edgeLabel["^ue", LightRed]],
    Property[Style[b → b, Dashed], EdgeLabels → edgeLabel["u", LightYellow]],
    Property[b → c, EdgeLabels → edgeLabel["e", LightGreen]],
    Property[c → d, EdgeLabels → edgeLabel["T:.", LightGreen]],
    Property[Style[d → e, Automatic],
      EdgeLabels → edgeLabel["L:.\nL > 0", LightGreen]],
    Property[Style[d → g, Automatic],
      EdgeLabels → edgeLabel["L:.\nL == 0", LightGreen]],
    Property[Style[g → h, Automatic],
      EdgeLabels → edgeLabel["B1:.", LightGreen]],
    Property[Style[g → fault, Thick],
      EdgeLabels → edgeLabel["invalid", LightRed]],
    Property[e → f, EdgeLabels → edgeLabel["F:.", LightGreen]],
    Property[Style[e → fault, Thick],
      EdgeLabels → edgeLabel["F ≤ 0", LightRed]],
    Property[f → f, EdgeLabels → edgeLabel["Σ < F\n&& Σ ≤ L", LightGreen]],
    Property[Style[f → e, Automatic],
      EdgeLabels → edgeLabel["F:\nΣ == F\n&& Σ ≤ L", LightGreen]],
    Property[Style[f → h, Automatic],
      EdgeLabels → edgeLabel["B1:.\nΣ == L+1", LightGreen]],
    Property[Style[h → exit, Automatic],
      EdgeLabels → edgeLabel["B2:.", LightGreen]],
    Property[Style[h → fault, Thick],
      EdgeLabels → edgeLabel["invalid", LightRed]],
    Property[Style[f → fault, Thick],
      EdgeLabels → edgeLabel["invalid", LightRed]],
    Property[Style[f → fault, Thick],
      EdgeLabels → edgeLabel["Σ > L+1", LightRed]]}],
  VertexSize → vs, ImageSize → is, GraphLayout → {gl, Center → fault}],
Grid[{{
  Style["Legend", "Text"], SpanFromLeft},
  {Style["Regular", "Text"], Style["Save input", "Text"]},
  {Style["Dashed", "Text"], Style["Discard input", "Text"]},
  {Style["Thick", "Text"], Style["Clear all saved", "Text"]},
  {Style["Dotted", "Text"], Style["No changes", "Text"]},

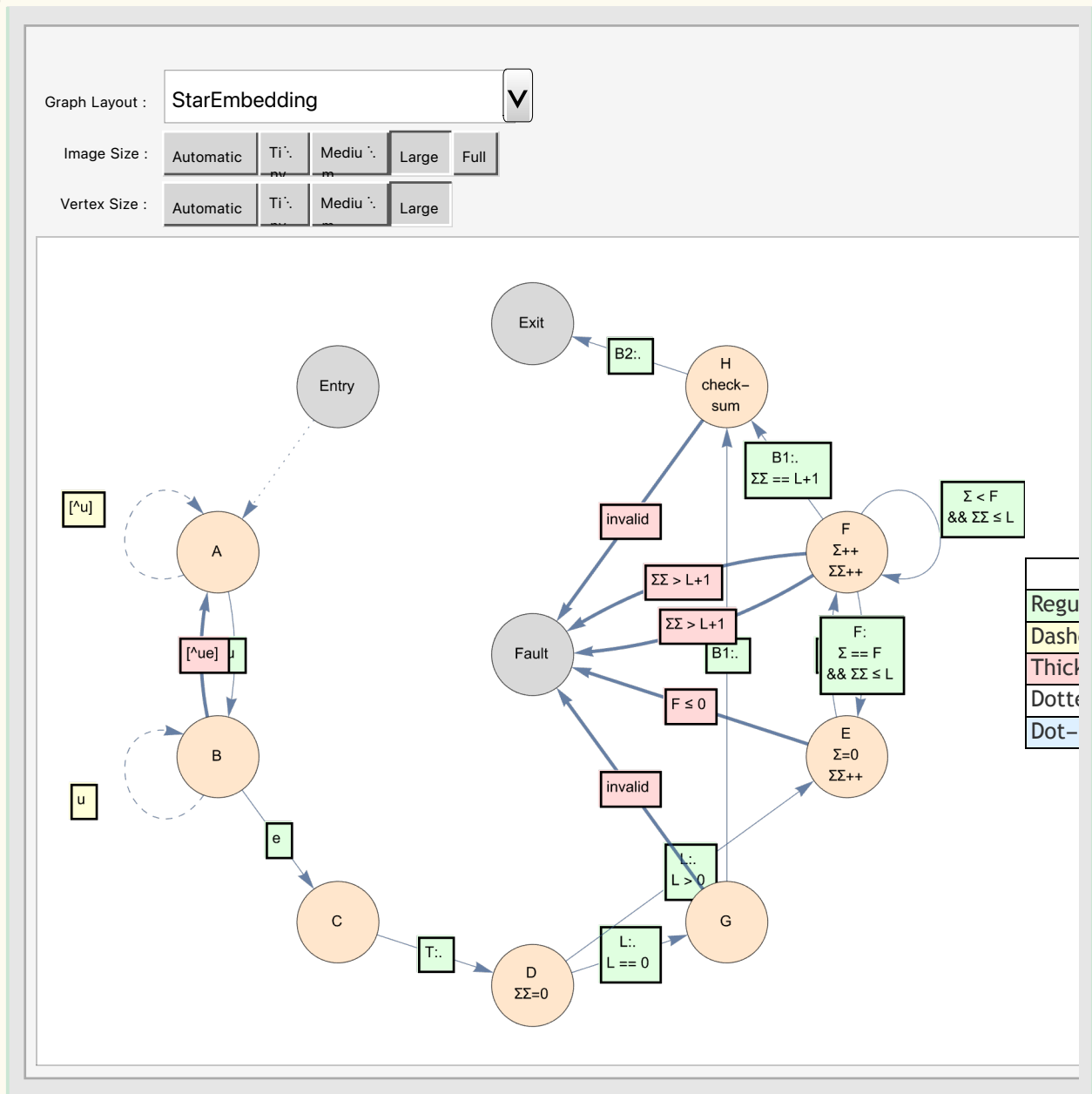
```

```

    {Style["Dot-Dash", "Text"], Style["Washback", "Text"]}},
    Frame → All, Alignment → {Left, Left, {1, 1} → Center},
    Background →
      {None, {White, LightGreen, LightYellow, LightRed, White, LightBlue}}]
  }]],
  {{gl, "StarEmbedding", "Graph Layout :"},
   Sort@{Automatic, "RadialEmbedding", "SpectralEmbedding", "PlanarEmbedding",
    "GridEmbedding", "LinearEmbedding", "BalloonEmbedding",
    "TutteEmbedding", "StarEmbedding", "CircularEmbedding",
    "DiscreteSpiralEmbedding", "SpiralEmbedding", "SpringEmbedding",
    "SpringElectricalEmbedding", "HighDimensionalEmbedding"}},
  {{is, Large, "Image Size :"}, {Automatic, Tiny, Medium, Large, Full}},
  {{vs, Large, "Vertex Size :"}, {Automatic, Tiny, Medium, Large}}]

```

Out[429]=

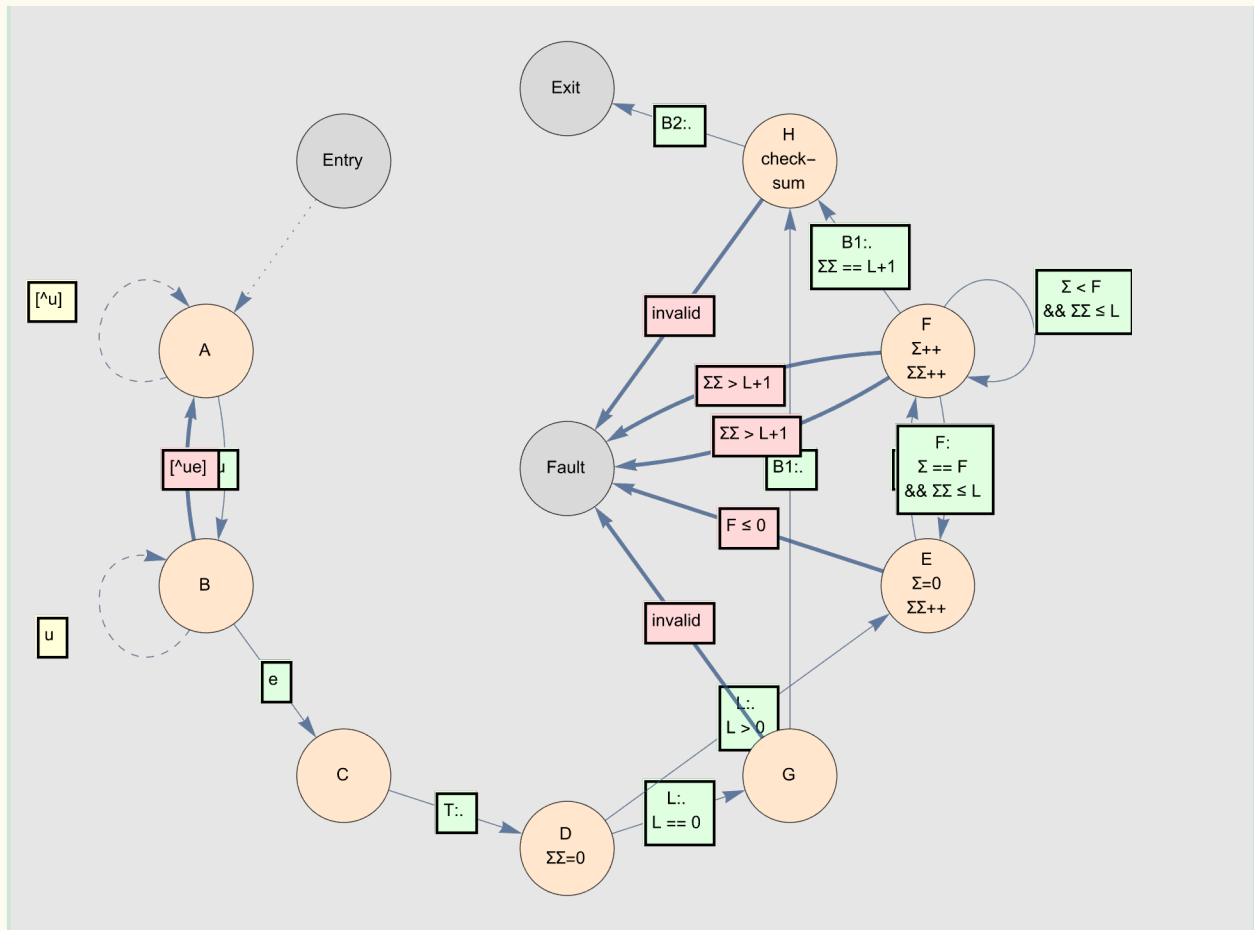



```

Property[b → c, EdgeLabels → edgeLabel["e", LightGreen]],
Property[c → d, EdgeLabels → edgeLabel["T:.", LightGreen]],
Property[Style[d → e, Automatic],
  EdgeLabels → edgeLabel["L:.\nL > 0", LightGreen]],
Property[Style[d → g, Automatic],
  EdgeLabels → edgeLabel["L:.\nL == 0", LightGreen]],
Property[Style[g → h, Automatic],
  EdgeLabels → edgeLabel["B1:.", LightGreen]],
Property[Style[g → fault, Thick],
  EdgeLabels → edgeLabel["invalid", LightRed]],
Property[e → f, EdgeLabels → edgeLabel["F:.", LightGreen]],
Property[Style[e → fault, Thick],
  EdgeLabels → edgeLabel["F ≤ 0", LightRed]],
Property[f → f, EdgeLabels → edgeLabel["Σ < F\n&& Σ ≤ L", LightGreen]],
Property[Style[f → e, Automatic],
  EdgeLabels → edgeLabel["F:\nΣ == F\n&& Σ ≤ L", LightGreen]],
Property[Style[f → h, Automatic],
  EdgeLabels → edgeLabel["B1:.\nΣ == L+1", LightGreen]],
Property[Style[h → exit, Automatic],
  EdgeLabels → edgeLabel["B2:.", LightGreen]],
Property[Style[h → fault, Thick],
  EdgeLabels → edgeLabel["invalid", LightRed]],
Property[Style[f → fault, Thick],
  EdgeLabels → edgeLabel["invalid", LightRed]],
Property[Style[f → fault, Thick],
  EdgeLabels → edgeLabel["Σ > L+1", LightRed]]],
VertexSize → Large, ImageSize → Large,
GraphLayout → {"StarEmbedding", Center → fault}]

```

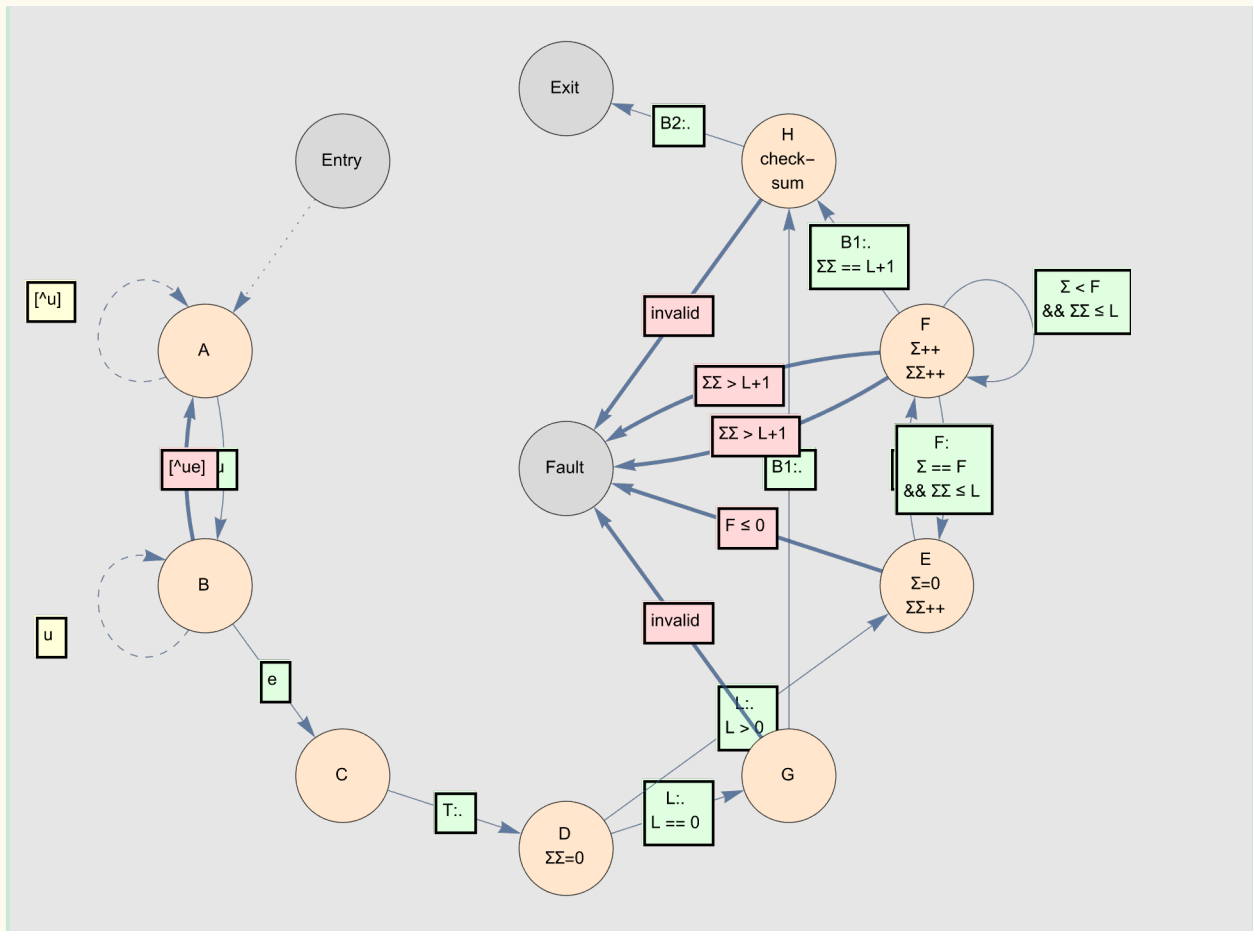
Out[430]=



In[431]:=

g\$

Out[431]=



In[432]:=

PropertyValue[{g\$, #}, VertexCoordinates] & /@ VertexList[g\$]

Out[432]=

```
{{-0.587785, 0.809017}, {-0.951057, 0.309017},
{-0.951057, -0.309017}, {-0.587785, -0.809017}, {6.04901 × 10-16, -1.},
{0.587785, -0.809017}, {0.951057, -0.309017}, {0.951057, 0.309017},
{0.587785, 0.809017}, {-7.04481 × 10-16, 1.}, {0., 0.}}
```

In[433]:=

VertexList[g\$] // InputForm

Out[433]//InputForm=

```
{"Entry", "A", "B", "C", "D\nΣΣ=0", "G", "E\nΣ=0\nΣΣ++",
"F\nΣ++\nΣΣ++", "H\ncheck-\nsum", "Exit", "Fault"}
```

In[434]:=

g\$ = Module[{init, a, b, c, d, e, f, g, h,

```

    exit, fault, vertices, vertexLabel, edgeLabel, colors},
vertexLabel = v  $\mapsto$  Placed[v, Center];
edgeLabel = {edge, bg}  $\mapsto$  Framed[edge, Background  $\rightarrow$  bg];
vertices =
  {init, a, b, c, d, g, e, f, h, exit, fault} =
    {"Entry", "A", "B", "C", "D\(\Sigma=0)", "G", "E\(\Sigma=0\)\(\Sigma++\)\(\SigmaF+=F)",
      "F\(\Sigma++\)\(\Sigma++\)", "H\(\check{check}-\)\(\Sigma\)", "Exit", "Fault"};
colors = <|"Entry"  $\rightarrow$  LightGray, "A"  $\rightarrow$  LightOrange,
  "B"  $\rightarrow$  LightOrange, "C"  $\rightarrow$  LightOrange, "D\(\Sigma=0)"  $\rightarrow$  LightOrange,
  "E\(\Sigma=0\)\(\Sigma++\)\(\SigmaF+=F)"  $\rightarrow$  LightOrange, "F\(\Sigma++\)\(\Sigma++\)"  $\rightarrow$  LightOrange,
  "G"  $\rightarrow$  LightOrange, "H\(\check{check}-\)\(\Sigma\)"  $\rightarrow$  LightOrange,
  "Exit"  $\rightarrow$  LightGray, "Fault"  $\rightarrow$  LightGray|>;
Graph[
  Property[Style[#, colors[#]], VertexLabels  $\rightarrow$  vertexLabel[#]] & /@ vertices,
  {Style[init  $\rightarrow$  a, Dotted],
    Property[Style[a  $\rightarrow$  a, Dashed],
      EdgeLabels  $\rightarrow$  edgeLabel["^u", LightYellow]],
    Property[a  $\rightarrow$  b, EdgeLabels  $\rightarrow$  edgeLabel["u", LightGreen]],
    Property[Style[b  $\rightarrow$  a, Thick], EdgeLabels  $\rightarrow$  edgeLabel["^ue", LightRed]],
    Property[Style[b  $\rightarrow$  b, Dashed], EdgeLabels  $\rightarrow$  edgeLabel["u", LightYellow]],
    Property[b  $\rightarrow$  c, EdgeLabels  $\rightarrow$  edgeLabel["e", LightGreen]],
    Property[c  $\rightarrow$  d, EdgeLabels  $\rightarrow$  edgeLabel["T:.", LightGreen]],
    Property[Style[d  $\rightarrow$  e, Automatic],
      EdgeLabels  $\rightarrow$  edgeLabel["L:.\nL > 1", LightGreen]],
    Property[Style[d  $\rightarrow$  g, Automatic],
      EdgeLabels  $\rightarrow$  edgeLabel["L:0", LightGreen]],
    Property[Style[d  $\rightarrow$  fault, Thick], EdgeLabels  $\rightarrow$  edgeLabel["L:1", LightRed]],
    Property[Style[g  $\rightarrow$  h, Automatic],
      EdgeLabels  $\rightarrow$  edgeLabel["B1:.", LightGreen]],
    Property[Style[g  $\rightarrow$  fault, DotDashed],
      EdgeLabels  $\rightarrow$  edgeLabel["!cs\nwb 3", LightBlue]],
    Property[e  $\rightarrow$  f, EdgeLabels  $\rightarrow$  edgeLabel["F:.", LightGreen]],
    Property[Style[e  $\rightarrow$  fault, DotDashed],
      EdgeLabels  $\rightarrow$  edgeLabel["F:0\nwb \(\Sigma+2)", LightBlue]],
    Property[Style[e  $\rightarrow$  fault, DotDashed],
      EdgeLabels  $\rightarrow$  edgeLabel["\(\SigmaF>L\)\nwb \(\Sigma+2)", LightBlue]],
    Property[Style[e  $\rightarrow$  fault, DotDashed],
      EdgeLabels  $\rightarrow$  edgeLabel["\(\Sigma>L\)\nwb \(\Sigma+2)", LightBlue]],
    Property[f  $\rightarrow$  f,
      EdgeLabels  $\rightarrow$  edgeLabel["PL:.\n\(\Sigma < F\)\n\(\Sigma \leq L)", LightGreen]],
    Property[Style[f  $\rightarrow$  e, Automatic],
      EdgeLabels  $\rightarrow$  edgeLabel["F:\n\(\Sigma == F\)\n\(\Sigma \leq L)", LightGreen]],
    Property[Style[f  $\rightarrow$  h, Automatic],
      EdgeLabels  $\rightarrow$  edgeLabel["B1:.\n\(\Sigma == L+1)", LightGreen]],

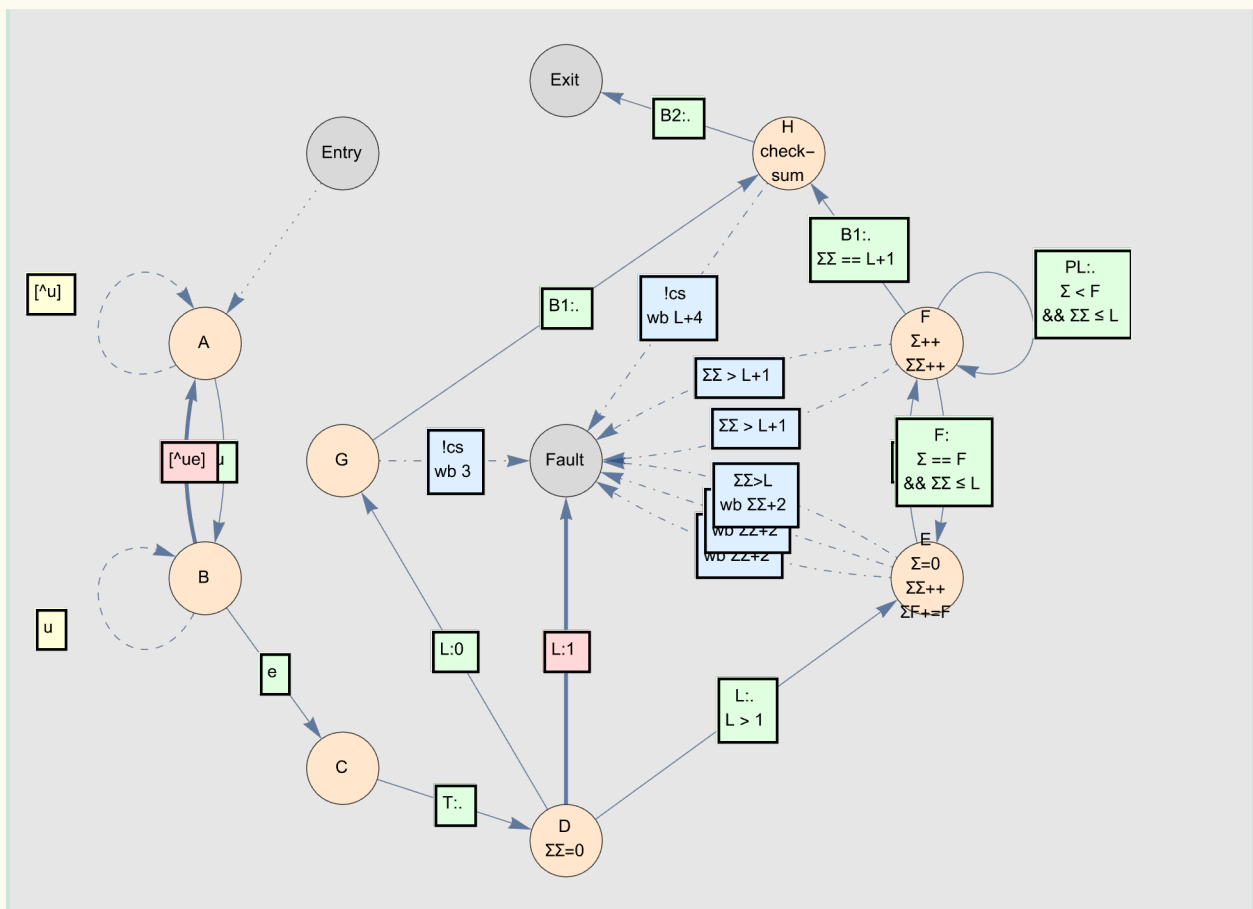
```

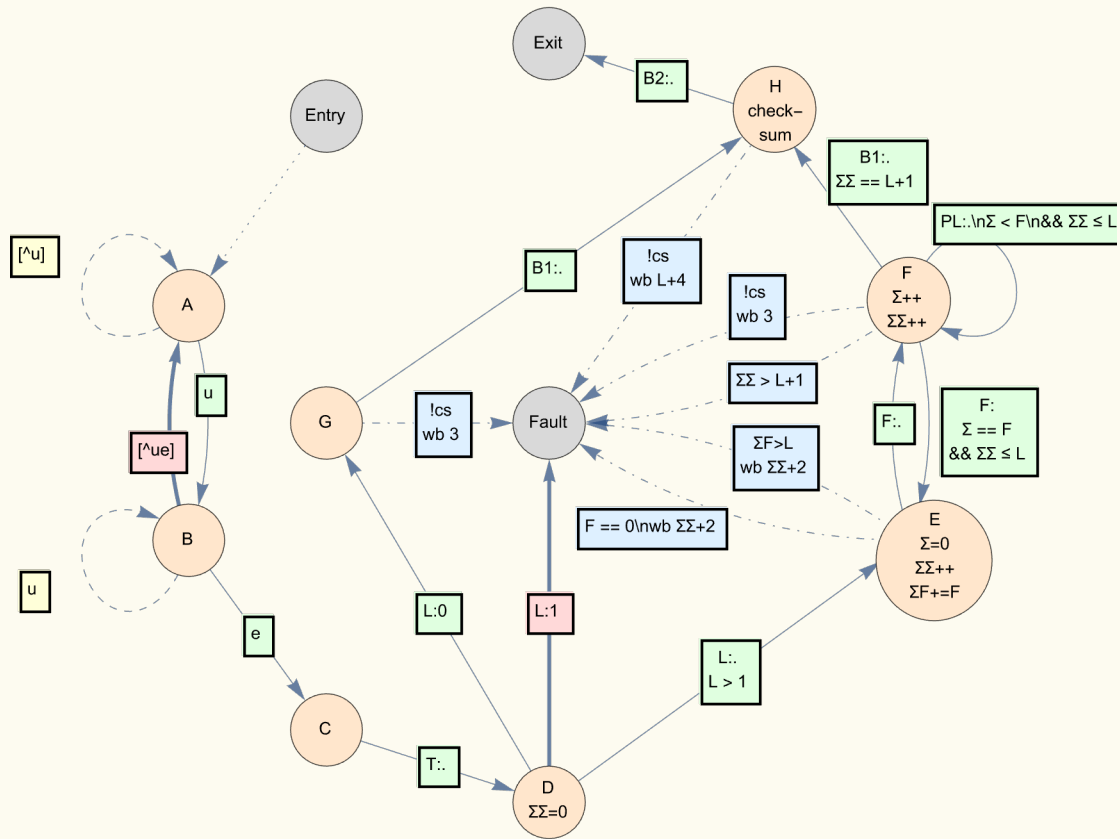
```

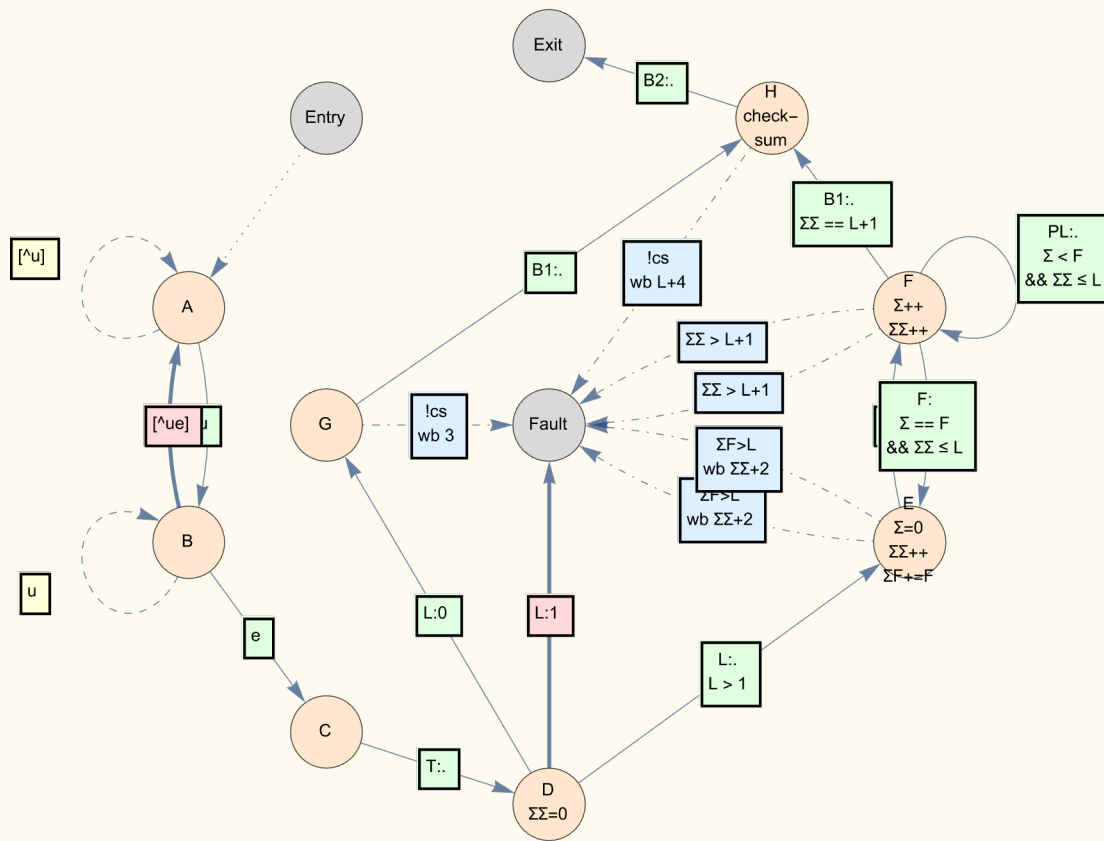
Property[Style[h  $\rightarrow$  exit, Automatic],
  EdgeLabels  $\rightarrow$  edgeLabel["B2:.", LightGreen]],
Property[Style[h  $\rightarrow$  fault, DotDashed],
  EdgeLabels  $\rightarrow$  edgeLabel["!cs\nwb L+4", LightBlue]],
Property[Style[f  $\rightarrow$  fault, DotDashed],
  EdgeLabels  $\rightarrow$  edgeLabel["invalid", LightBlue]],
Property[Style[f  $\rightarrow$  fault, DotDashed],
  EdgeLabels  $\rightarrow$  edgeLabel[" $\Sigma\Sigma > L+1$ ", LightBlue]]},
VertexSize  $\rightarrow$  Large, ImageSize  $\rightarrow$  Large,
VertexCoordinates  $\rightarrow$  {{-0.5877852522924737`, 0.809016994374948`},
  {-0.9510565162951538`, 0.3090169943749484`},
  {-0.9510565162951534`, -0.30901699437494645`},
  {-0.5877852522924726`, -0.8090169943749468`}, {0, -1.`},
  {-0.5877852522924738`, 0}, {0.9510565162951539`, -0.3090169943749485`},
  {0.9510565162951533`, 0.30901699437494634`},
  {0.5877852522924726`, 0.8090169943749468`}, {0, 1.`}, {0., 0.`}}]]

```

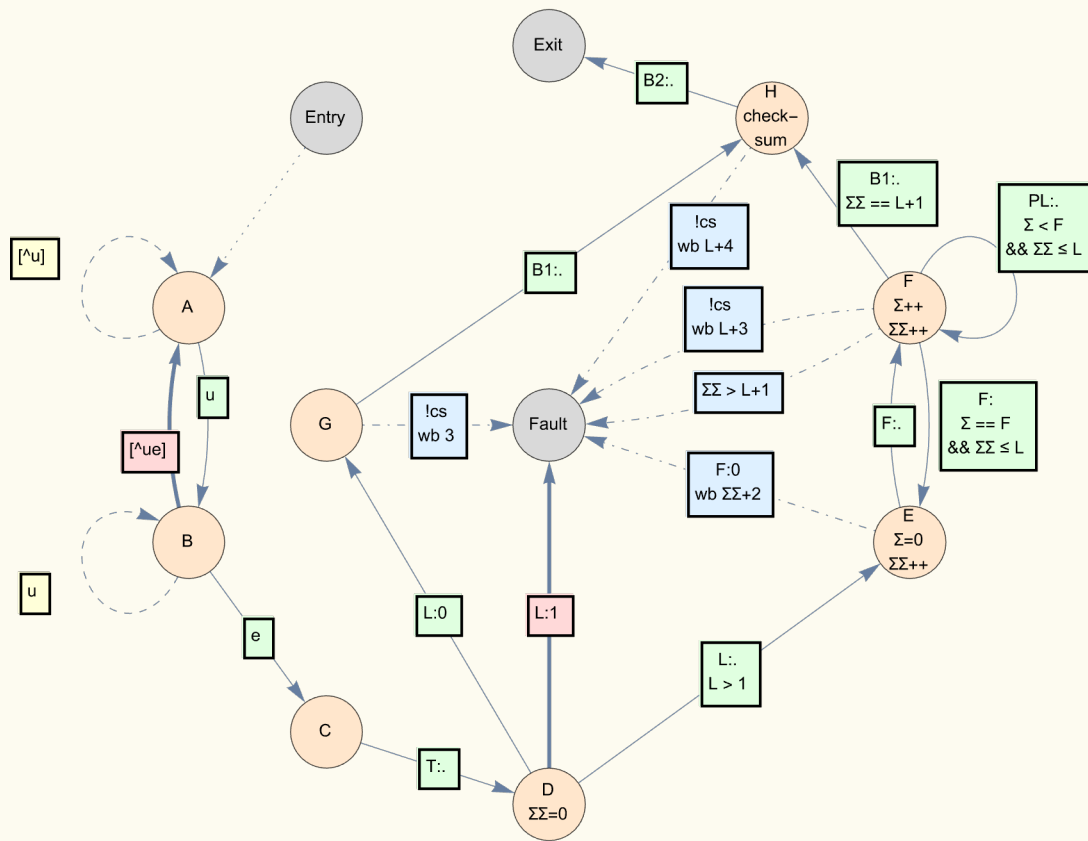
Out[434]=

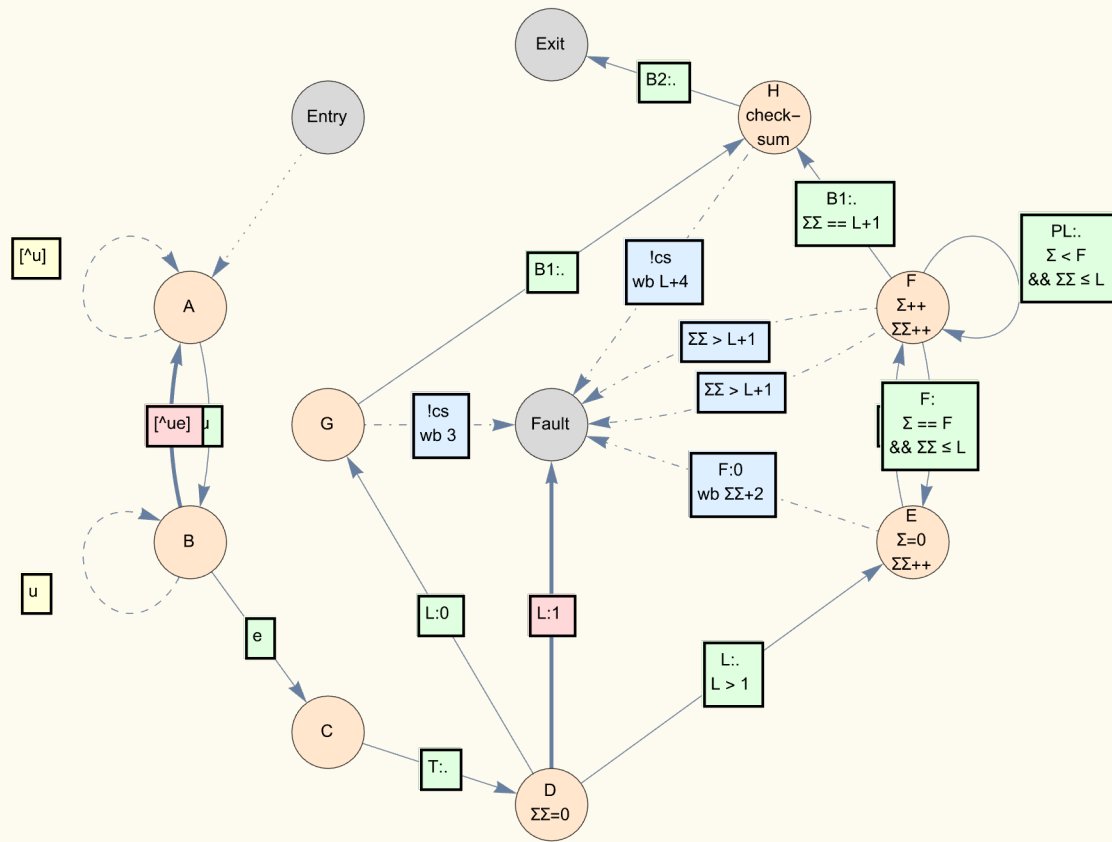


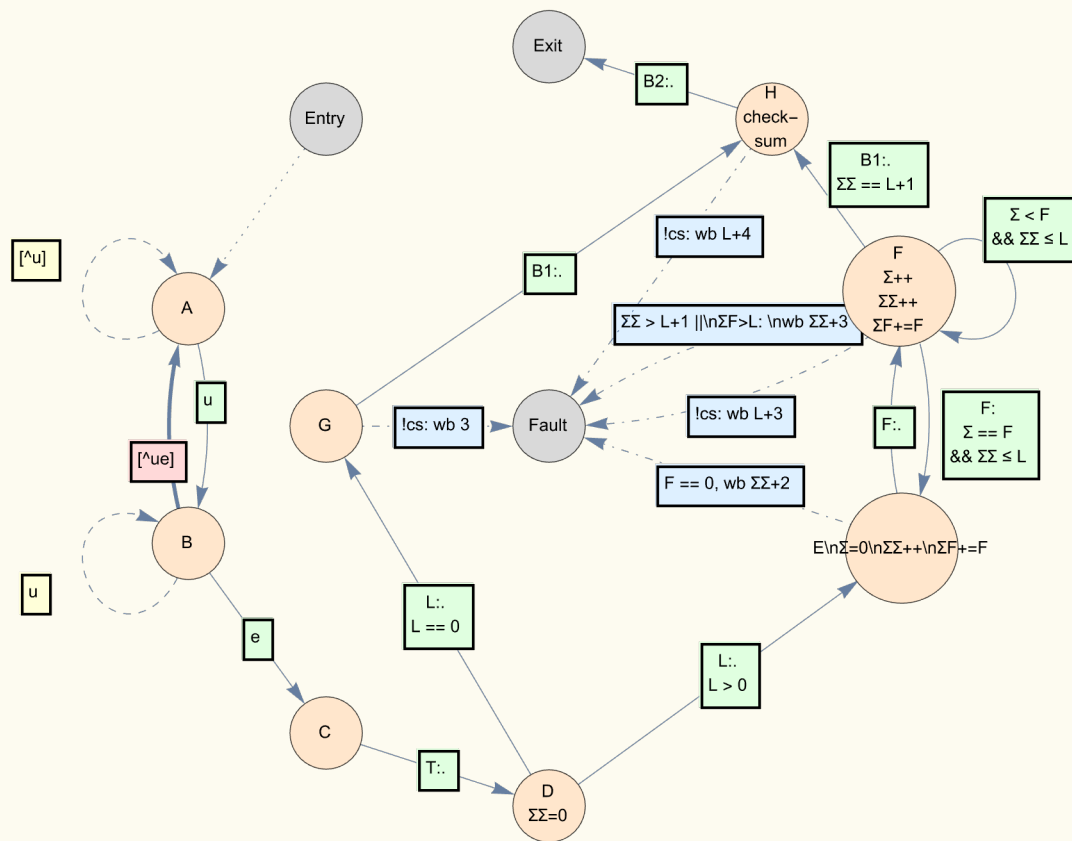


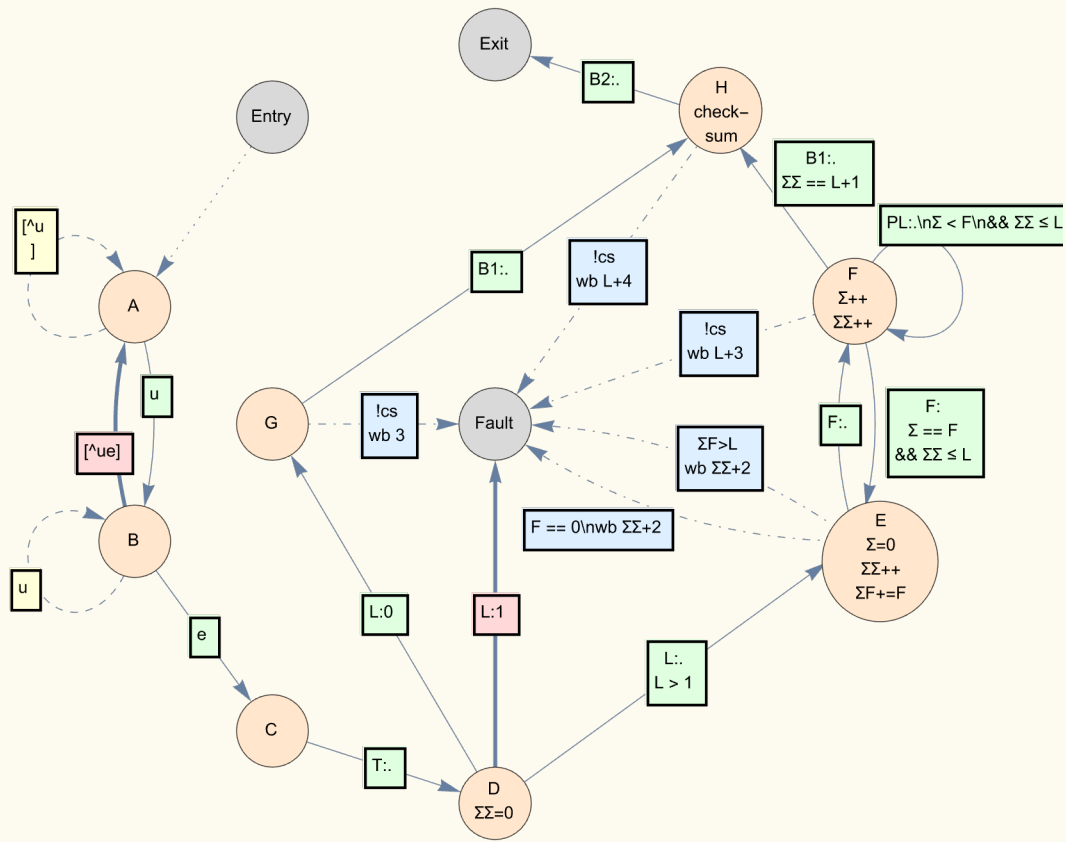


 Set: Tag Inherited in Inherited [State] is Protected.









Legend	
Regular	Save input
Dashed	Discard input
Thick	Clear all saved
Dotted	No changes
DotDash	Washback