

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 chances 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 correct sequence of bytes and can restart in the midst of certain kinds of noise. Such a scanner is an alternative to more general *error detection and correction* (EDAC) for, say, inexpensive systems that furnish a checksum at most.

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*.

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 handle noisy prefixes and can rewind 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 a corrupted header leading to misinterpreted payload bytes. In this case, the scanner pushes back all the misinterpreted characters, discard the deceptive header bytes, and restart.

For instance, suppose we want to recognize the sequence ue^{TL} , where u and e are literal characters, 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 regex 'dot,' meaning "any character."

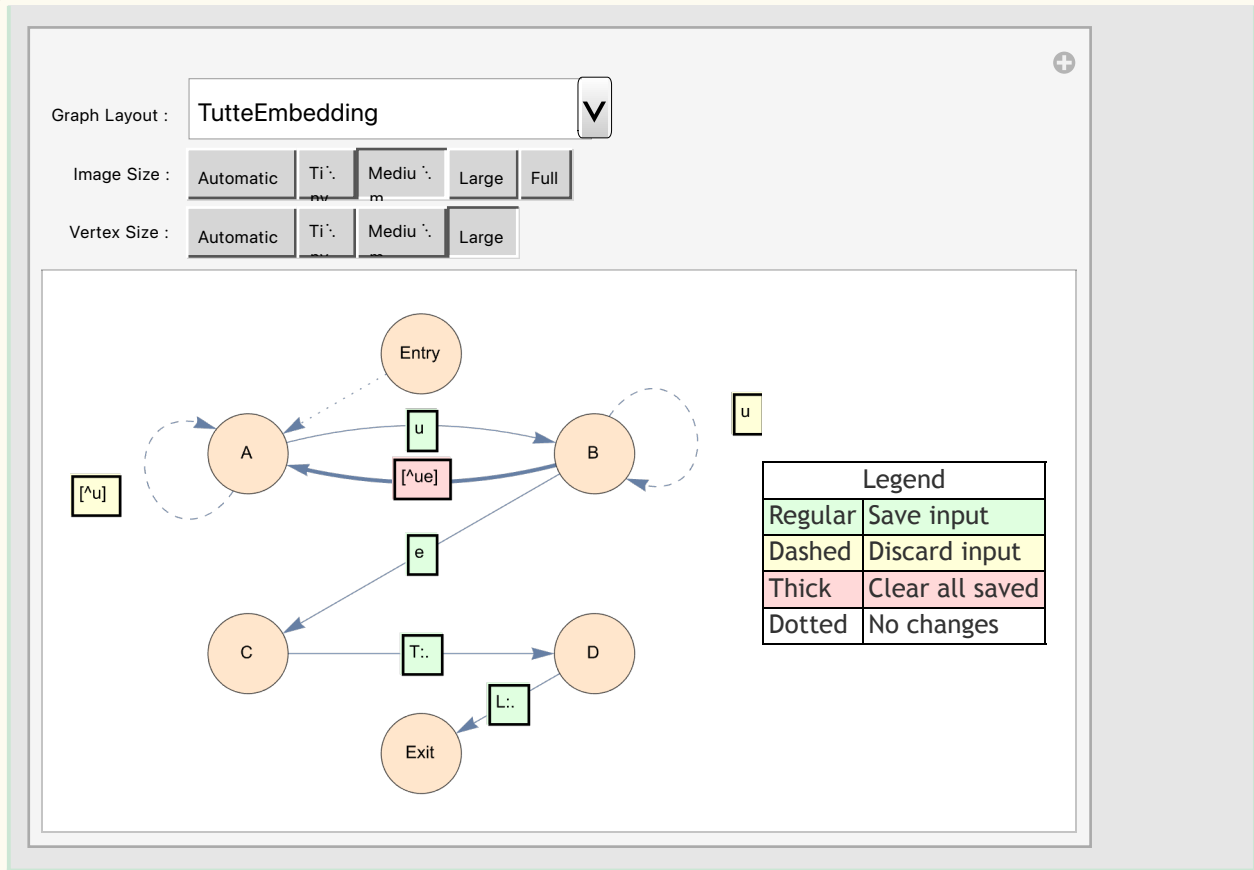
In[622]:=

```

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[622]=



Definitions

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 scanner-state buffers from an input buffer. Instances of the *Circular Buffer abstract data type* described below control the scanner and input states 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. Scanners must discard invalid characters.

■ Circular Buffer

Restartable Scanners

The system repeatedly interrupts the application with one or more new input bytes from an input stream, say a peripheral device. More concretely, the interrupt service routine, ISR, receives (a pointer to) the caller-maintained input buffer and a length, the number of characters transferred to the system from the peripheral. The ISR does not know the capacity of the caller-supplied input buffer: it's the responsibility of the application to manage communication with the peripheral, to ensure there is enough room so that input buffers are never overflowed. In practice, static bounds may be available from the peripheral's manufacturer, or a tuning phase before deployment may be necessary to establish practical or heuristic bounds.

Algorithm

A scanner and two FSMs are detailed below. The scanner models its current state with a circular buffer (CB) called *scannerState*, models partially validated results with another buffer called *provisionalPayload*, and wraps the input with another circular buffer called *inputState*. The scanner calls its FSM one character at a time. The FSM returns action specifications and a “next vertex” indicator to the scanner. The scanner performs the action to modify the buffers, fetches a new character, and calls the FSM at the “next vertex” if there is a new character.

The first FSM just processes headers and can correct certain kinds of errors. The second FSM processes payload bytes and can wash partially validated results back to the input to handle a broader class of errors. Both FSMs can be interrupted at any time and restarted later. The scanner will replay validated characters from *scannerState* to bring the FSM to its prior vertex, then continue with new input. When the input is exhausted, the scanner stops in its last-visited vertex after performing the specified action.

INVARIANT: *scannerState* only ever contains valid characters, either newly validated or already validated in a prior pass.

On a call or reentry, the scanner restarts its FSM at its unique *Entry* vertex and saves the current head index of *scannerState*. The reason is that the scanner may pop valid characters from *scannerState* for replay. Those characters are still valid and must be retained when the scanner resumes processing input at the prior last-visited vertex.

The scanner maintains two metrics: *discarded* and *saved*. 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 to the scanner state while taking regular edges with the “Save input” label. While taking thick edges with the “Clear all saved” label, the *discarded* metric is incremented by the length of the scanner state, which is then cleared, and the *saved* metric is set to zero. “Washback” edges are described later. They do not appear in the FSM above, which only recognizes headers.

The scanner and FSMs maintain a dictionary called *matches* for edges labeled with meta-variables *T* and *L*. This dictionary facilitates lookup of those metadata by name.

As usual when designing C code with *Mathematica*, we make temporary variables in a *Module* for mutable data, 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 variables and structs 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* is the same as the type of its first parameter. Both are Mathematica *Lists* in curly braces: prior vertex, action command, a parameter for the action, and the current *matches* dictionary. The FSMs ignore the input action and action parameter and pass back new actions and parameters to the scanner, which performs them. The returned actions are "save," "discard," "clear," "washback," or "no action."

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

A Stepping FSM for the Header

The following FSM is a straightforward transcription of the diagram at the beginning of this document. After replaying any saved scanner state, the scanner calls this FSM in vertex *Entry*. There is no action to perform in vertex *Entry*, so the FSM returns immediately to the scanner. The scanner takes a character from the input, if one is available, and unconditionally transitions to vertex *A*. If no character is available, the scanner stops in vertex *Entry*.

If the incoming character to vertex *A* is **u**, the FSM returns to the scanner with a *save* action and a transition indicator to vertex *B*. The scanner performs the action and optimistically collects a new character. If a character is available, the scanner calls the FSM at vertex *B* with that character. If, instead, the incoming character to vertex *A* is anything other than **u**, *A* returns a *discard* action and a transition back to *A*. The scanner performs the action and optimistically collects a new character. If a character is available, the scanner calls the FSM at vertex *A* again, or stops in vertex *A* if there is no new character. Staying in *A* handles arbitrary noise before a **u**.

Vertex *B* tells the scanner to save an incoming **e** character and transition to vertex *C*, to discard an incoming **u** character and stay in *B*, and to discard all saved characters and transition to *A* otherwise. Vertex *B* handles both stuttering of **u**, a common case, and a sporadic **u**, not followed by **e**.

This design cannot handle stuttering of **e**. It has no choice but to treat **e** after **e** as a type byte, the value of *T*. This limitation is inherent in the design of the data stream from the peripheral device—it gives us no guidance for validating *T*. If we had such guidance, we could easily encode it in the FSM. Likewise for vertex *D*. It cannot detect an invalid *L* character. It has no choice but to save it. In the next FSM, where we consume payload, we'll address validating *L* by counting payload bytes.

In[686]:=

```
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* emulates the FSM caller in a real system. It is responsible only for managing the buffers. It gets and performs action commands from *fsmStep*, but 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}.

Let's write and test the scanner. It's important to read some of the tests because they illustrate the metric difference between saving characters from input versus replaying them from *scannerState*.

■ Support Routines

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

The caller, *Scanner*, furnishes three CBs that are modified by side effect. The three CBs are *scannerState*, *provisionalPayload*, and *inputState*.

Scanner is foldable in that its output type is the same as the type of its first input, but can't be called from Fold because characters may wash back from provisional payload to the input. We have not experimented with a Fold that modifies the stream being folded over.

□ Action *save*

The action *save* with argument *validated* increments the metric for validated characters and pushes the character to *scannerState*. *Save* with argument *provisional* a character into *provisionalPayload* whence it can be washed back when a fault is detected.

□ Action *clear*

The action *clear* ignores its argument, increments the metric for discarded characters by the number of characters in *scannerState*, sets the metric for saved characters to zero, and empties out *scannerState*. This action does not modify *provisionalPayload* or *inputState*.

□ Action *discard*

The action *discard* ignores its argument, increments the metric for discarded characters by one, and does not modify buffers.

□ Action *washback*

The action *washback* transfers a number of characters from *provisionalPayload* back to *inputState*. The number of characters is specified by the argument passed to *washback*. It does not modify metrics because it does not manipulate saved or discarded characters; it just puts them back into the input. The FSM above does not call *washback*, but the new one, below, does.

□ Scanner itself

This scanner first replays all validated characters in *scannerState* to handle restart or re-entry. Any action but *save* during replay is a catastrophic error (*Throw*, useful only in simulation; some other method of handling catastrophic errors must be implemented in a real, embedded application).

Scanner performs actions before optimistically reading characters from *inputState* and transitioning to the next vertex. When *inputState* is empty, the scanner stops.

In[695]:=

```
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"] += CBlen[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_, (*pattern that matches anything*)
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 the First FSM and Scanner

■ Support Routines

In[697]:=

```

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

```

In[699]:=

```
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[701]:=

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

In[702]:=

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

Queues are displayed in the order *scannerState*, containing validated characters, *provisionalPayload*, and *inputState*. For this FSM, we should never see provisional characters. Metrics are displayed first, then the metadata dictionary containing values for *T* and *L*, if any.

■ No Input

Expect to stop in vertex *Entry* with empty queues.

In[704]:=

```
expect[testScanner["", ""],
  {<|"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" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"discarded" → 0, "saved" → 0|>, <| |>}]
```

```
<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
```

Out[704]=

```
PASS
```

■ A Single, Invalid Input Character

Expect to stop in vertex *A* with one discarded and no saved characters. The character is visible in *inputState*, but the head and tail pointers have advanced.

In[705]:=

```
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|>, <|>}]
```

```
<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <|>, new_matches → <|>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <|>, new_matches → <|>|>
```

Out[705]=

PASS

■ A Single, Valid Input Character

Expect to stop in vertex *A* with one saved character and no discarded characters.

In[706]:=

```
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|>, <|>}]
```

```
<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <|>, new_matches → <|>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <|>, new_matches → <|>|>
```

Out[706]=

PASS

■ A Complete, Valid Sequence

Expect to stop in vertex *D* with four saved, no discarded characters, and the input pointers advanced to position 4.

In[707]:=

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

```
<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new_matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new_matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new_matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new_matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new_matches → <|T → 74, L → 75|>|>
```

Out[707]=

PASS

■ Incomplete Scanner State

Expect to stop in vertex *A* with NO saved characters; vertex *A* is achieved by replay from *scannerState*.

In[708]:=

```
expect[testScanner["u", ""],
  {<|"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" → {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    "head" → 0, "tail" → 0|>, <|"discarded" → 0, "saved" → 0|>, <| |>}]
```

```
<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new_matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new_matches → <| |>|>
```

Out[708]=

PASS

■ Stuttering

Expect to stop in vertex *D* with 6 discarded and 4 saved characters.

In[709]:=

```
expect[testScanner["", "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" → 6, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]
```

```
<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new matches → <|T → 74, L → 75|>|>
```

Out[709]=

PASS

Expect to stop in vertex *D* with 7 discarded and 3 saved characters. The first **u** is replayed from *scannerState* and not saved again. The remaining 7 **u** characters are taken from input.

In[710]:=

```
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|>}]
```

```
<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <|>, new matches → <|>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <|>, new matches → <|>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <|>, new matches → <|>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <|>, new matches → <|>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <|>, new matches → <|>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <|>, new matches → <|>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <|>, new matches → <|>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <|>, new matches → <|>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <|>, new matches → <|>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <|>, new matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new matches → <|T → 74, L → 75|>|>
```

Out[710]=

PASS

■ Scanner State with Ignored Valid Characters

Expect 2 saved, with $T == x == 120$ and $L == y == 121$. The start signal `ue` is replayed from *scannerState* and not saved again.

In[711]:=

```
expect[testScanner["ue", "xyzabcdef"],
  {<|"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, 0}, "head" → 2, "tail" → 9|>,
    <|"discarded" → 0, "saved" → 2|>, <|"T" → 120, "L" → 121|>}]
```

```
<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <|>, new matches → <|>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <|>, new matches → <|>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <|>, new matches → <|>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <|>, new matches → <|T → 120|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 120|>, new matches → <|T → 120, L → 121|>|>
```

Out[711]=

PASS

Expect 1 saved, with $T == X == 88$, $L == Y = 89$, and 3 replayed.

In[712]:=

```
expect[testScanner["ueX", "YZABCDEF"],
  {<|"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|>}]
```

```

<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new matches → <|T → 88|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 88|>, new matches → <|T → 88, L → 89|>|>

```

Out[712]=

PASS

Expect 4 saved and 4 discarded. The initial, replayed **u** causes retraction to vertex *A* from vertex *B*, where the saved queue is cleared.

■ A Little Junk in the Input

In[713]:=

```

expect[testScanner["", "uasdfueJK"],
  {<|"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, 97, 115, 100, 102, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0},
    "head" → 9, "tail" → 9|>, <|"discarded" → 4, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]

```

```

<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → clear, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new matches → <|T → 74, L → 75|>|>

```

Out[713]=

PASS

Expect 4 saved and 4 discarded. The initial, replayed **u** causes retraction to vertex *A* from vertex *B*, where the saved queue is cleared.

In[714]:=

```

expect[testScanner["u", "asdfueJK"],
  {<|"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|>}]

```

```

<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → clear, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new matches → <|T → 74, L → 75|>|>

```

Out[714]=

PASS

Expect 1 discarded and 4 saved.

In[715]:=

```

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|>}]

```

```

<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → clear, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new matches → <|T → 74, L → 75|>|>

```

Out[715]=

PASS

Expect 2 discarded and 4 saved.

In[716]:=

```

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, 0},
    "head" → 6, "tail" → 6|>, <|"discarded" → 2, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]

```

```

<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → clear, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new matches → <|T → 74, L → 75|>|>

```

Out[716]=

PASS

Expect 3 discarded and 4 saved.

In[717]:=

```

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, 0},
    "head" → 7, "tail" → 7|>, <|"discarded" → 3, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]

```

```

<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → clear, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new matches → <|T → 74, L → 75|>|>

```

Out[717]=

PASS

Expect 4 discarded and 4 saved.

In[718]:=

```

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|>}]

```

```

<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → clear, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new matches → <|T → 74, L → 75|>|>

```

Out[718]=

PASS

■ Lots of Junk in the Input

Expect 11 discarded and 4 saved.

In[719]:=

```

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, 117, 88,
      88, 88, 117, 117, 101, 74, 75, 0}, "head" → 15, "tail" → 15|>,
    <|"discarded" → 11, "saved" → 4|>, <|"T" → 74, "L" → 75|>}]

```



```

<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → clear, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → discard, arg → no arg,
  newVertex → A, matches → <| |>, new matches → <| |>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → discard, arg → no arg,
  newVertex → B, matches → <| |>, new matches → <| |>|>
<|oldVertex → B, action → save, arg → validated,
  newVertex → C, matches → <| |>, new matches → <| |>|>
<|oldVertex → C, action → save, arg → validated,
  newVertex → D, matches → <| |>, new matches → <|T → 74|>|>
<|oldVertex → D, action → save, arg → validated,
  newVertex → Exit, matches → <|T → 74|>, new matches → <|T → 74, L → 75|>|>

```

Out[719]=

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[720]:=

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

```
<|oldVertex → Entry, action → no action,
  arg → no arg, newVertex → A, matches → <|>, new matches → <|>|>
<|oldVertex → A, action → save, arg → validated,
  newVertex → B, matches → <|>, new matches → <|>|>
```

Out[720]=

PASS

■ Results

In[721]:=

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

Out[721]=

```
<|success count → 17, 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 → {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 → {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}, head → 0, tail → 0|>,
    <|cap → 16, origin → {0, 0, 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, 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, 0},
    head → 0, tail → 0|>, <|cap → 16,
    origin → {117, 117, 117, 117, 117, 117, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 10, tail → 10|>, <|discarded → 6, 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, 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, 0},
    head → 0, tail → 0|>, <|cap → 16,
    origin → {117, 117, 117, 117, 117, 117, 117, 101, 74, 75, 0, 0, 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, 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, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {120, 121, 122, 97, 98, 99, 100, 101, 102, 0, 0, 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, 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, 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, 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, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 97, 115, 100, 102, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0},
    head → 9, tail → 9|>, <|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, 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, 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, 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, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {88, 117, 101, 74, 75, 0, 0, 0, 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, 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, 0}, head → 0, tail → 0|>,
    <|cap → 16, origin → {117, 88, 117, 101, 74, 75, 0, 0, 0, 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, 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, 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 →
  {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, 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 → {}|>

```

Scanning More

The scanner and FSM above accept only a header containing a start signal **ue**, a type byte, *T*, and a length byte, *L*.

The next, full FSM also recognize the header, but continue to collect payload and checksum bytes from the input. The length byte *L* gives us the entire length of validated payload characters, *excluding* the four bytes of the header **ueTL**, in particular *excluding L itself*. A packet **ueTL** with no payload is allowed; *L* can be zero.

If *L* is non-zero, the downstream data consists of one or more non-empty *fields*. Each field, *F*, contains a *relative index byte* followed by content bytes. The relative index byte of a field, also called *F*, equals the length of the field in bytes, *including F itself*. Because a field may not be empty, *F* must be at least 2, and the sum of all *F*s in the payload must be at least 2. We have the following invariants to check: *F* cannot be zero; *L* cannot be one; the sum of all relative index bytes *F* must equal the total length byte, *L*.

The Full FSM

The full FSM maintains running variables, C_1 , C_2 , for the Fletcher checksum, initialized on entry, as are all the other variables, Σ , $\Sigma\Sigma$, ΣF , *T*, *L*, *F*, B_1 , and B_2 . The variables Σ , $\Sigma\Sigma$, and ΣF track the lengths of each field, the length of all the fields, and the sum of all *F* bytes seen. The running checksum bytes, C_1 and C_2 ,

are checked against the consumed checksum bytes, B_1 and B_2 , generating faults if either does not match. The detailed logic follows.

In *Entry*, initialize variables, take no action, collect a character, and transition to vertex *A* with that character.

In vertex *A*, if the character received is `u`, save it as *validated*, accumulate checksum bytes C_1, C_2 , collect a new character, and transition to vertex *B*. Otherwise, discard the character and stay in vertex *A*.

If the input character to vertex *B* is `u`, stay in *B* to handle stuttering of `u` (a common case). If the input character is `e`, save it as *validated*, accumulate checksum bytes C_1, C_2 , collect a character, and transition to vertex *C*. Otherwise, clear input and validated chars (*scannerState*), collect a character, and transition back to *A*, handling a garbled `ue` prefix.

In vertex *C*, unconditionally take the received character as the type byte *T*, save it as *validated*, accumulate checksum bytes C_1, C_2 , collect a character, and transition to vertex *D*.

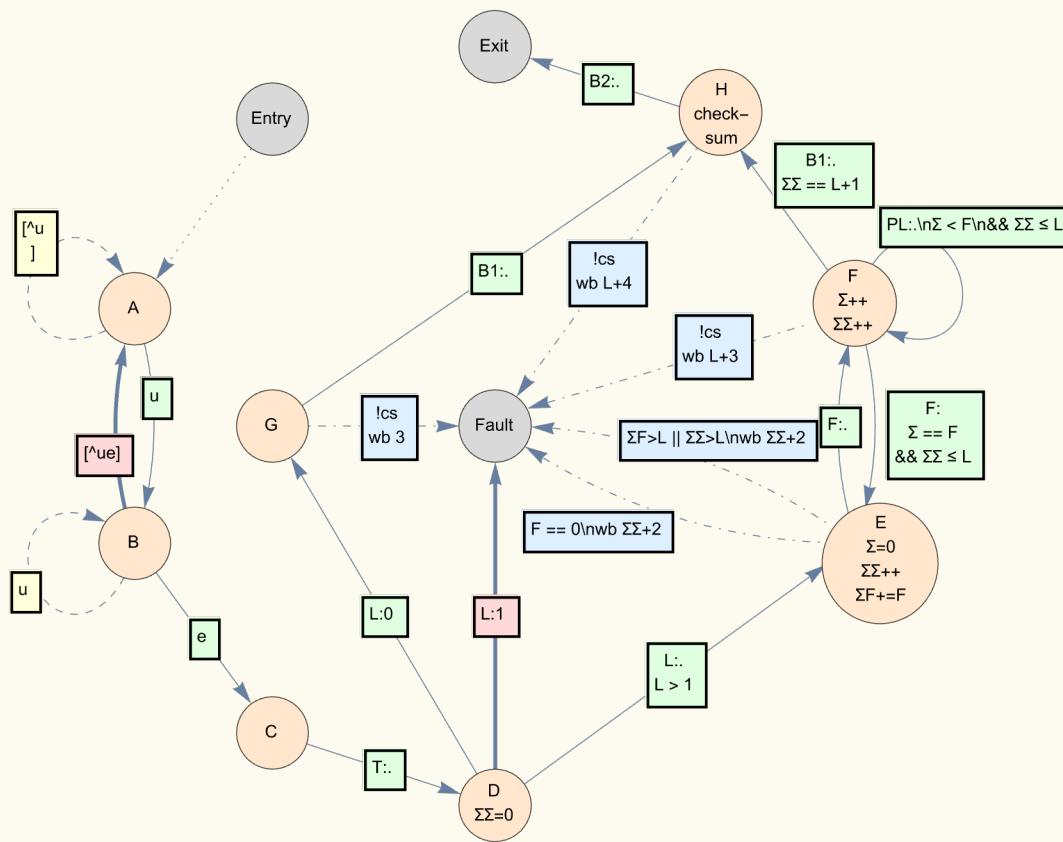
In vertex *D*, initialize Σ , the sum of all field bytes (including their relative index bytes) to zero. Unconditionally, take the received character as the cumulative number of payload bytes, *L*. If $L == 1$, clear all validated characters and transition to fault state for later restart. If $L == 0$, save it as *validated* and transition to vertex *G* to collect checksum bytes. If $L > 1$, save *L* as *validated*, accumulate checksum bytes C_1, C_2 , collect a character as the next relative index byte or field-length byte *F*, and transition to vertex *E*.

Vertex *E* is all about processing field lengths. Initialize $\Sigma = 0$, increment Σ by one to account for the received *F*, and increment ΣF by *F*. If $F == 0$, we have received a bad *F* from vertex *D*. Wash back $\Sigma \Sigma + 2$ (all bytes collected so far, including the current *F* and the upcoming payload byte *P* that was collected optimistically) and transition to *Fault*. Likewise if $\Sigma F > L$, the total length of all fields, meaning that the received *F* was too big, or if $\Sigma \Sigma > L$, another signal that we've received more characters than expected. If all goes well, save *F* as provisional, accumulate checksum bytes C_1, C_2 , collect a payload byte *P* and transition to vertex *F*.

Vertex *F* is all about processing payload bytes. Collect $F - 1$ payload bytes, accumulating checksum bytes C_1, C_2 , and incrementing and testing Σ and $\Sigma \Sigma$. When $\Sigma == F$ and $\Sigma \Sigma \leq L$, we've consumed the final payload byte of the current field. Transition back to vertex *E* with a byte for the next field length *F*. If $\Sigma \Sigma == L + 1$, the byte under consideration is probably the first checksum byte, B_1 ; transition to *H*. 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 possible cases:

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

■ Not Enough Field Data

char	vertex	buffers	Σ	ΣΣ	Σ < F && ΣΣ ≤ L	Σ == F && ΣΣ ≤ L	ΣΣ == L + 1	ΣF	ΣF > L ΣΣ > 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	ueT32ppB ₁ B ₂	2	3	false	true	false	2		3	2		
x	E	ueT32ppB 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	ueT34ppppB ₁ B ₂		0					3				
F : 4	E	ueT34ppppB ₁ B ₂	0	1				4	3	4			
PL : p	F	ueT34ppppB ₁ B ₂	1	2	true	false	false	4	3	4	p		
PL : p	F	ueT34ppppB ₁ B ₂	2	3	true	false	false	4	3	4	p		
PL : p	F	ueT34ppppB ₁ B ₂	3	4	false	false	true	4	3	4		p	
B ₂ : B ₁	H	ueT34ppppB 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	ueT34ppppB ₁ B ₂		0			3				
F : 4	E	ueT34ppppB ₁ B ₂	0	1	4	true	3	4			

■ Fletcher Checksum

https://en.wikipedia.org/wiki/Fletcher%27s_checksum

In[722]:=

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

■ newFsmStep

In[724]:=

```
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_}, (* accumulator of the fold *)
    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;
     $\Sigma$ ++;
     $\Sigma$ 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[ $\Sigma == F \&\& \Sigma \Sigma \leq L$ ,
    (matches["F"] = F = character;
    {C1, C2} = fletcher[{C1, C2}, character];
    {"E", "save", "provisional"}),
    ((*character identified as payload byte*)
    If[ $\Sigma < F \&\& \Sigma \Sigma \leq L$ ,
      (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

Wrap these in `Block[{Print=Identity}, ...]` to suppress verbose tracing.

■ Old Cases

□ No Input

In[726]:=

```
expect[testScanner["", "", 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" → {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 → <| |>|>
```

Out[726]=

PASS

□ A Single, Invalid Input Character

In[727]:=

```
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[727]=

PASS

□ A Single, Valid Input Character

In[728]:=

```
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[728]=

PASS

□ A Complete, Valid Sequence

Type T is 74, length L is 75 (that's echoed in the output):

In[729]:=

```

expect[testScanner["", "ueJK", 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, 101, 74, 75, 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[729]=

PASS

□ Incomplete Scanner State

In[730]:=

```
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[730]=

PASS

□ Stuttering

In[731]:=

```
expect[testScanner["u", "uuuuuuueJK", 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, 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[731]=

PASS

□ Scanner State with Payload Characters

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

In[732]:=

```
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[732]=

```
{<|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|>|>
```

In[733]:=

```
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}, "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[733]=

PASS

In[734]:=

```

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[736]:=

```
Block[{Print = Identity}, dpyScanner["ue", "xyzabcdef", newFsmStep]]
```

Out[736]=

```
***** 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[737]:=

```
Block[{Print = Identity}, dpyScanner["ueJ", "KZABCDEFG", newFsmStep]]
```

Out[737]=

```
***** 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	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	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:	75	90	65	66	67	68	69	70	0	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[738]:=

```
Block[{Print = Identity},
  expect[testScanner["ueJ", "KZABCDEF", 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" →
      {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"|>}}]]
```

Out[738]=

PASS

□ A Little Junk in the Input

In[739]:=

```
Block[{Print = Identity}, dpyScanner["u", "asdfueJK", newFsmStep]]
```

Out[739]=

```
***** 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[740]:=

```
Block[{Print = Identity},
  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|>}}]
```

Out[740]=

PASS

In[741]:=

```
Block[{Print = Identity}, dpyScanner["u", "uXXuueJK", newFsmStep]]
```

Out[741]=

```
***** 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[742]:=

```
Block[{Print = Identity}, 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|>}}]]
```

Out[742]=

PASS

□ Lots of Junk in the Input

In[743]:=

```
Block[{Print = Identity}, dpyScanner["u", "uuuuuuuXXXuueJK", newFsmStep]]
```

Out[743]=

```
***** 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[744]:=

```
Block[{Print = Identity}, expect[testScanner["u", "uuuuuuuXXXuueJK", 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, 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[744]=

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[745]:=

```
Block[{Print = Identity}, dpyScanner["u", "uuuuuuuXXXXuueJK", newFsmStep]]
```

Out[745]=

```
***** 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[746]:=

```
Block[{Print = Identity},
  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|>, <| |>}]}
```

Out[746]=

PASS

□ Results

In[747]:=

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

Out[747]=

```
<|success count → 29, 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 → {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 → {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 → 6, 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, 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, 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, 0}, head → 0, tail → 0|>,
<|cap → 16, origin → {117, 97, 115, 100, 102, 117, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0},
  head → 9, tail → 9|>, <|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 → {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 → {88, 117, 101, 74, 75, 0, 0, 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, 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, 117, 101, 74, 75, 0, 0, 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, 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, 101, 74, 75, 0, 0, 0, 0, 0, 0, 0, 0, 0},

```



```

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, 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[748]:=

```

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

```

In[750]:=

```

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[752]:=

```

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[754]:=

```
Block[{Print = Identity},
  dpyScannerWithNumbers[{117, 101, 0, 255}, {}, newFsmStep]]
```

Out[754]=

***** 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[755]:=

```
Block[{Print = Identity},
  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|>}}]
```

Out[755]=

PASS

□ Correctly parses a valid packet

In[756]:=

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

Out[756]=

***** 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[757]:=

```
Block[{Print = Identity},  
  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},  
      "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|>}}]]
```

Out[757]=

PASS

□ With Bookend Partitioning

In[758]:=

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

Out[758]=

***** 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	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	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:	198	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:	198	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[759]:=

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

Out[759]=

***** 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[760]:=

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

Out[760]=

***** 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[761]:=

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

Out[761]=

***** 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[762]:=

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

Out[762]=

```
{102, 167}
```

In[763]:=

```
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[763]=

```

***** 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[764]:=

```
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[764]=

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

In[765]:=

```
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[765]=

```
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