Emitables and the Mechanics of Potential Sequences

Grant Haywood

Draft 12/6/2023 #1

Definition of Terms

The Sequence

“In mathematics, a sequence is an enumerated collection of objects in which repetitions are allowed and order matters. Like a set, it contains members (also called elements, or terms). The number of elements (possibly infinite) is called the length of the sequence. Unlike a set, the same elements can appear multiple times at different positions in a sequence, and unlike a set, the order does matter. Formally, a sequence can be defined as a function from natural numbers (the positions of elements in the sequence) to the elements at each position. “

The Observer

“In information theory, any system which receives information from an object”

The Emitable

The Emitable is a Recursive Information Packet.
An opaque wrapper for an arbitrary chunk of information, it has one function:

Function<Observer Stream<Emitable>> emit

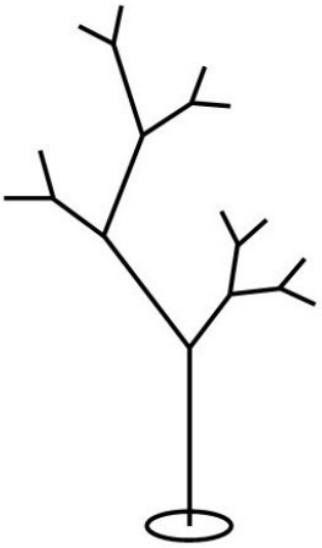
or in traditional Java notation:

Stream<Emitable> emit(Output observer);

The emit function will write its data, if any, to the output. That is to say the output Observes the content of the Emitable Information Packet. This operation permutes the state of the output, and as such emit is not a pure function.
When called, emit returns a stream of 0 or more additional Emitables.
In this way the Emitable is a recursive data type.

“In computer programming languages, a recursive data type (also known as a recursively-defined, inductively-defined or inductive data type) is a data type for values that may contain other values of the same type. Data of recursive types are usually viewed as directed graphs”
-- https://en.wikipedia.org/wiki/Recursive_data_type

An Emitable can create a 1 to N mapping, and its children can also create 1 to N[c] mappings, where N[c] is potentially unique to each child Emitable.



Sympodial branching

“Sympodial growth is a bifurcating branching pattern where one branch develops more strongly than the other, resulting in the stronger branches forming the primary shoot and the weaker branches appearing laterally.”

Flow Control Patterns

The Iterator and Time:
Iterators are defined in terms of 2 functions:

Boolean hasNext()
T next()

hasNext() indicates the Iterator has more T elements to produce, and next() retrieves one of these T elements.

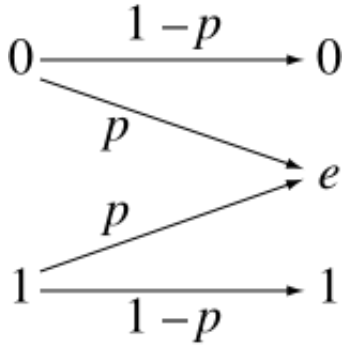
The upshot is that this does implement a very convenient form of flow control. However it has a serious downside. hasNext() is actually an implicit promise. A promise that there will be another T available, whenever next() is called. You have provided no information to hasNext() about when next will be called. The other issue is that information propagates. You could share your hasNext() result with your friend, running in another thread. If your friend also has access to this Iterator, and call next(), without telling you, the promise your call to hasNext() made is no longer valid, without coordination, you have no clue how long the hasNext() result will be valid for.

Streams and next():
Java Streams allow the user to build up a thread safe chain of functional transformations.
> stream chain example
Streams however prohibit talking about individual items in the traditional imperative way.

Enter the Optional

An Optional represents a Value that is either present or missing. Missing information can be an indicator, the absence is the information being conveyed.

When elements of a Sequence are wrapped in an Optional, the absence of value can be used to indicate the end of the Sequence.
This Optional Sequence pattern is equivalent to an erasure channel It maps an Unary input to a Binary output
V → V || e



“A binary erasure channel (BEC) with erasure probability p is a binary input, ternary output channel. The possible channel outputs are 0, 1, and a third symbol 'e' called an erasure. The erasure represents complete loss of information about an input bit. The capacity of the BEC is 1 − p bits per channel use.”

The Potential Sequence

Optional<T> getNext();

The Potential Sequence introduces a new flow control pattern that is in some sense half way between the Iterator and the Stream. This is a thread safe construct upon which one can build functional chains of operations, but unlike the Java Stream, retains the ability for the programmer to work with individual items in the sense of next().

An empty result may indicate the end of a sequence, or simply a gap in the sequence, depending on how a Potential Sequence is being used.
Using Optional.empty() as an indicator for the end of the sequence allows one to retain the functionality of hasNext() while fusing the next() and hasNext() functions into a single operation.
By wrapping each element in an Optional, the next element in a Potential Sequence can be derefrenced safely at any time.

Sequence Mechanics:

* many of these constructs can be implemented with Streams, so this is could also be termed Stream Mechanics

Cycle:
A Sequence that repeats until some indicator has been reached

Pinion System:
A system of 2 cycles, or gears, that meet at an intersection point

Slip Wheel:
A pinion system where the gear analogy is relaxed and an element on wheel A can meet multiple elements on wheel B

Gear Box:
A collection of Pinion Systems whos outputs are flatMaped onto a single output stream

Collapse:
A Potential Sequence is converted to a Sequence by unwrapping each element and removing empty values.