Quid2 Binary (First Draft)

Pasqualino 'Titto' Assini (tittoassini@gmail.com)

11th of November 2015

Contents

For more information and the most recent version of this specification check http://quid2.org.

Quid2 Bits

Quid2 Binary is an universal data encoding system.

Design Goals

The universality of an encoding system can be measured along at least three different coordinates: - Expressivity: capacity of faithfully representing data structures. - Space: suitability for information systems of any level of complexity (nanosystems to mainframes). - Time: suitability for long term storage of data.

- Simple to implement. A single encoding/decoding rule. No need to support a range of 'primitive types' unless they are needed by your system.
- Efficient. Fast and optimal bit-oriented encoding.

Quid2 Binary aims to score highly on the first two? / three dimensions by adopting these design principles:

Quid2 Bits is a principled, simple and efficient data encoding .

of values of known data types.

rather than being based on a fixed set of predefined data structures, Quid2 Bits can encode values of any user-defined data type.

- Discovered, not invented. Based on the simplest possible abstractions and free from arbitrary and limiting decisions regarding supported data structures or primitive types.

These goals have led to an unusual design that is quite different from most existing data interchange standard: no built-in data structures or primitive types of any kind, bit-encoded rather than byte-encoded and

Design Non-Goals:

Bit-encoding:

Different architectures use different word sizes (though currently they are all 2-powers of bytes).

The bit is the only primitive and non-arbitrary unit of information.

For transmission/storage bits sequences need to be mapped to words but this is external to the system.

Quid2 binary defines a 1-N mapping between values of any algebraic data type, as defined in Quid2 Model, and binary sequences.

Definitions

A data type is composed by a name and a constructors tree.

A constructors tree is a balanced, right-heavy, binary tree whose leaves are constructors and whose internal nodes have a left and right arrow, respectively marked with a `0` and a `1` bit.

A constructor is composed by a name (unique in the data type) and a (possibly empty) sequence of fields each of which points to a data type.

A value is a

Examples

The figure shows some simple data types, with one (`Unit`), two (`Bool`) or five (`N`) constructors with no fields.

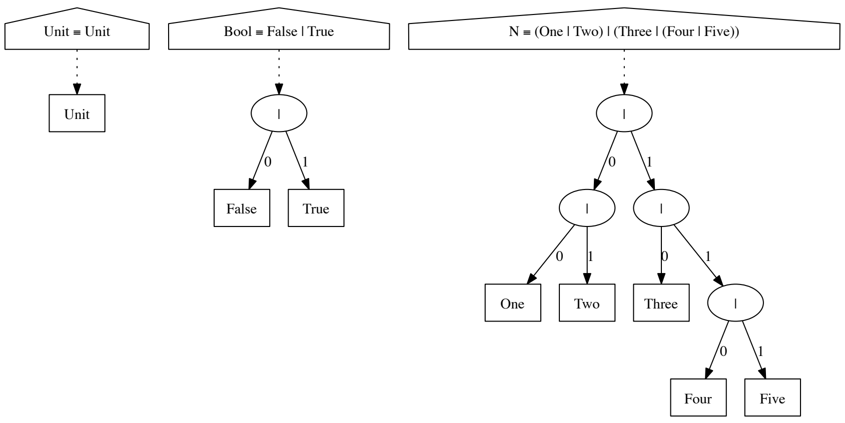

Figure 1: Simple types

Note how the 5 constructors of N are split in groups of 2 and 3 (balanced and right-heavy) and the right group of 3 constructors is split in groups of 1 and 2.

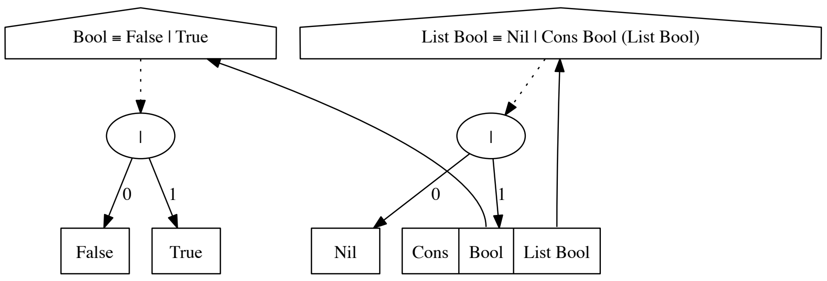

Figure 2: A List of Booleans

A more complex example, a list of booleans.

A `List` is either a `Nil` (a 0-length list) or a `Cons` constructor with two fields, the first pointing to a `Bool` and the second to the `List Bool` itself.

We can also have infinite (codata) data types.

A `Bool` is necessarily finite, a `List` can be be finite, Stream is by definition infinite.

Being infinite, Stream cannot be a static data structure, both and encoder and a decoder are processes that either produce or consume a bit stream.

The set of values of Bool .. is finite, that of values of List Bool (any list?) is countable, what about a Stream Bool

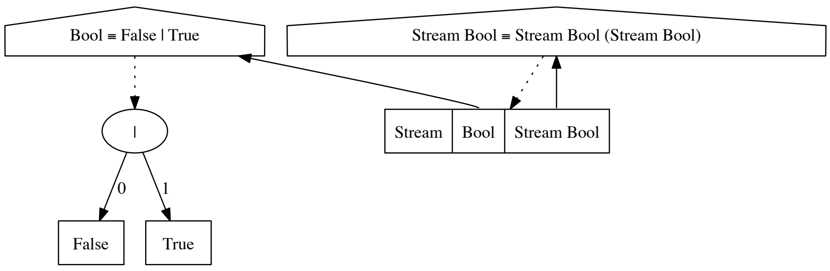

Figure 3: A Stream of Booleans

Table 1: Boolean values and codes

Value	Code
False	0
True	1

Table 2: N values and codes

Value	Code
One	00
Two	01
Three	10
Four	110
Five	111

Table 3: List values and codes

Value	Code
Nil	0
Cons True Nil	110
Cons False (Cons True Nil)	10110

Coding Rule

The code of a value is the sequence of bits obtained by joining left-to-right the markers on the path from the data type to the value constructor, followed by the code of the values in the constructors' fields.

Data types with single constructor have 0-length codes.

Optimal encoding: takes as little space as possible assuming that all encoded sequences are equi-probable.

Complete encoding: there are no erroneous codes, if the decoder asks for one more bit it can always interpret it.

Multiple

-- Example, see file:///Users/titto/workspace/quid2/tests/tree5.svg data L = A | B | C | D | E

Optimised Encoding and Decoding

Conveniently, when the number of constructor is a power of two, this bit encoding is equivalent to the usual unsigned encoding, see file:///Users/titto/workspace/quid2/tests/tree8.svg

A smart encoder/decoder will just write/read a byte to encode/decode a Word8, no `tree traversal` necessary. * No assumption on word-size: bit-encoding rather than byte or word-encoding.