Student: João Pedro Marçal Storino (B2)
Teacher: Maxime Lefrançois
Majeur: Informatique
Subject: Data Interoperability and Semantics
Date: 22/10/2024

Exam 01

Part 1 - (5 points) Comparison of main data formats.

On one large full-page table, shortly describe or compare each data formats from List 1 in terms of the features and criteria from List 2.

NOTE: Be concise. You don't need to justify.

List 1. Data Formats:

1. CSV
2. JSON
3. XML
4. YAML

List 2. Features and Criteria

1. **Simplicity:** How easy is the format to create, read, and modify by both humans and machines?
2. **Human-Readability:** Can the data be easily understood and edited in a text editor or similar tool?
3. **Interoperability:** Is the format widely supported across different tools, software, and platforms?
4. **File Size/Overhead:** How lightweight is the format in terms of file size, especially for large datasets?
5. **Metadata Support:** Does the format allow embedding of metadata (e.g., column types, descriptions, units) within the file?
6. **Structural Complexity:** Can the format handle complex data structures, such as nested, hierarchical, relational, or graph data?
7. **Query:** Are there dedicated query language for the format?
8. **Schema Validation:** Does the format support a schema or mechanism to enforce data consistency and integrity?
9. **Extensibility:** Can the format be extended with custom structures or elements without breaking compatibility (e.g., custom tags or schemas)?
10. Can the format be extended with custom structures or elements without breaking compatibility (e.g., custom tags or schemas)?

Answer:

Features/Criteria	CSV	JSON	XML	YAML
1. Simplicity	+	=	−	+
2. Human-Readability	+	=	−	+
3. Interoperability	+	+	+	=
4. File Size/Overhead	+	=	−	+
5. Data Type Support	−	+	+	+
6. Metadata Support	−	=	+	=
7. Structural Complexity	−	+	+	+
8. Query	−	=	+	−
9. Schema Validation	−	=	+	=
10. Extensibility	−	+	+	+

Part 2. (/20 points) ASN.1

ASN.1 is a standard interface description language for defining data structures that can be serialized and deserialized in a cross-platform way. It provides a formal way to describe data and enables interoperability across different systems by ensuring consistent data encoding and decoding. While ASN.1 is now 40 years old, it is still broadly used in telecommunication and computer networking, such as for 5G mobile phone communications, LDAP directories, Securing HTTP communications with TLS (X.509) Certificates, Intelligent Transport Systems, and the Interledger Protocol for digital payments.

Protocol developers define data structures in ASN.1 modules, which are generally a section of a broader standards document written in the ASN.1 language. Here are some common ASN.1 base data types:

BOOLEAN [tag number 0110]: value can be TRUE or FALSE

INTEGER [tag: 0210]: a signed integer. A valid range can be specified with the notation (min..max)

BIT STRING [tag number 0310]: used for bit arrays, where each bit has an individual meaning.

ENUMERATED [tag number 1010]: a list of named items.

SEQUENCE [tag number 1610]: a collection of items to group together.

CHOICE [n/a]: one of the items can be present at a time.

IA5String [tag number 2210]: a printable ASCII string.

Below is an ASN.1 module definition, adapted from the ETSI Intelligent Transport Systems (ITS) Common Data Dictionary definition. You can find more details at the following link: `https://forge.etsi.org/rep/ITS/asn1/cdd_ts102894_2`.

Note: "`ego`" is how we name the vehicle on which the communicating ITS system is deployed. "`alter`" is another vehicle that is observed by "`ego`", or that communicates with "`ego`".

```
ETSI-ITS-CDD DEFINITIONS AUTOMATIC TAGS
::= BEGIN
EgoData ::=SEQUENCE { --invented for the
    Exam --
    id              IA5String,
    energyStorage   EnergyStorageType,
    speed           SpeedValue,
    driveDirection  DriveDirection,
    lanePosition    LanePositionOptions
}
AlterData ::= SEQUENCE { -- invented --
    id              IA5String OPTIONAL,
    message         IA5String OPTIONAL,
    safeDistance    SafeDistanceIndicator,
    speed           SpeedValue,
    driveDirection  DriveDirection,
    lanePosition    LanePositionOptions
}
EnergyStorageType ::= BIT STRING { -- real
     def --
    hydrogenStorage       (0),
    electricEnergyStorage (1),
    liquidPropaneGas      (2),
    compressedNaturalGas  (3),
    diesel                (4),
    gasoline              (5),
    ammonia               (6)
}(SIZE(7))
SafeDistanceIndicator::= BOOLEAN -- real
    def --
SpeedValue ::= INTEGER { -- unit is 0,01 m
    /s --
    standstill     (0),
    outOfRange     (16382),
    unavailable    (16383)
} (0..16383)
DriveDirection ::= ENUMERATED { -- real
    def --
    forward    (0),
    backward   (1),
    unavailable (2)
}
LanePositionOptions ::= CHOICE { -- real
    def --
    simplelanePosition LanePosition,
    simpleLaneType LaneType,
    detailedlanePosition
        LanePositionAndType,
    ...
}
LanePositionAndType::= SEQUENCE { -- real
    def --
    transversalPosition LanePosition,
    laneType LaneType DEFAULT traffic,
    ...
}
LanePosition ::= INTEGER { -- real def --
    offTheRoad        (-1),
    innerHardShoulder (0),
    outerHardShoulder (14)
} (-1..14)
LaneType::= INTEGER{ -- simplified: only
    some
    values            --
    traffic           (0),
    pedestrian        (12),
    parking           (17),
    emergency         (18)
    ...
}(0..31)
END
```

Because ASN.1 is both human-readable and machine-readable, an ASN.1 compiler can compile modules into libraries of code, codecs, that decode or encode the data structures.

ASN.1 defines different encoding rules that specify how to represent a data structure as bytes. Basic Encoding Rules (BER) is the oldest one, Packed Encoding Rules (PER) is the most compact. XML Encoding Rule (XER) is based on XML. JSON encoding rules (JER) is the easiest to start playing with ASN.1 and to debug applications.

In this part we will consider two messages, one about "ego", one about "alter":

Message 1: "Ego is a Highway Grass Cutting Machine with id AB-123-CD. It uses and stores diesel, liquid propane gas, and electricity. It drives forward on the outer hard shoulder at a speed of 10.8 km/h."

Message 2: "Alter EF-456-GH is moving forward at 144 km/h on traffic lane number 3, not respecting safe distances"

Note: Check out the appendices A-E, as they are all important to answer the questions in this part.

Question 1. (1 pt) Justify that the encoded value for speed 10.8 km/h is integer 300.

Answer:
We have that the speed given is 10.8 km/h. Tranfsorming in m/s we have:

$$\frac{10.8}{3.6} = 3 \text{ m/s}$$

The system represents speed in units of 0.01 m/s , meaning that each integer represents 0.01 m/s. So:

$$3\,\text{m/s} = 300 \times 0.01\,\text{m/s}$$

Thus, the speed value 10.8 km/h corresponds to the encoded value **300** in the system.

Question 2. (2 pts)

- How IEEE 754 floating point number would encode the value 144.0 ?

Answer:
We know that 144 is a integer number and, in binary is represented by:

$$10010000_2$$

Thus, representing in $1.bits \times 2^{exp}$, we have:

$$10010000_2 = 1.0010000 \times 2^7$$

Thus, the value of the exponent (e) is:

$$e = 7$$
$$7 + 127 = 134$$

And, the value of 134_2 in binary is:

$$1000\ 0110_2$$

Now, for the Mantissa, we have the value after the explicit 1. Remember that it should have 23 bits. So:

$$001\ 0000_2$$

and, with 23 zeros,

$$001\ 0000\ 0000\ 0000\ 0000\ 0000$$

Since the number is positive, the value of the signal is zero. Thus, the fial representation in IEEE 754 is:

$$0\ 10000110\ 00100000000000000000000$$

Where:
0: represents the sign bit,
10000110: represents the exponent,
00100000000000000000000: represents the mantissa.

And, in hexadecimal:

$$\underbrace{0100}_{4}\ \underbrace{0011}_{3}\ \underbrace{0001}_{1}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}$$

Resulting in: 0x43100000.

- Find an IEEE 754 floating point number that approximates 10.8 at ±0.05

Answer:
First of all, since is impossible to convert 10.8, we are going to use a range of ± 0.5. In this exercise, lets consider our number as 10.75. So, transforming 10.75 in binary:

We have that the integer part is 10 and the fractional part is 0.75. The integer part is represented by:

$$10_{10} = 1010_2$$

For the fractional part, we have to do the following opperations:

$$\text{Multiply } 0.75 \times 2 = 1.5$$

Record 1 for the integer part, and, after:

$$\text{Multiply } 0.5 \times 2 = 1.0$$

Record 1 for the integer part, and, since we've achieved the 0, we stop. Therefore, the number 10.75 in binay is:

$$1010.11$$

Normalizing the binary, we have:

$$1.01011 \times 2^3$$

Since the number is positive, the value of the sign is 0 and the expoent value is 130, which is:

$$1000\ 0010_2$$

The mantissa, is equal to 01011 with zeros until 23 bits.

$$010\ 1100\ 0000\ 0000\ 0000\ 0000_2$$

So, the final IEEE 754 number is equal to:

$$0\ 10000010\ 01011000000000000000000$$

Where:
0: represents the sign bit,
10000010: represents the exponent,
01011000000000000000000: represents the mantissa.
And, in hexadecimal:

$$\underbrace{0100}_{4}\ \underbrace{0001}_{1}\ \underbrace{0010}_{2}\ \underbrace{1100}_{C}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}$$

Resulting in: 0x412C0000.

Question 3. (4 pts) Write a document that could be a plausible JER encoding of Message 1 about "ego"

Answer:

```
{
    "EgoData": {
      "id": "AB-123-CD",
      "energyStorage": "10110", // diesel(4), liquidPropaneGas(2), electricEnergyStorage
          (1) -> BIT STRING
      "speed": 1080, // 10.8 km/h is 3 m/s -> 3 m/s = 300 * 0.01 m/s -> SpeedValue = 300
      "driveDirection": 0, // 0 = forward
      "lanePosition": {
        "simplelanePosition": 14 // outerHardShoulder
      }
    }
  }
```

Question 4. (1 pt) BER-encode the BIT STRING. `diesel+liquidPropaneGas+electricEnergyStorage`

Answer:
For the purpose of encoding, we need to first define each feature (like diesel, liquidPropaneGas, and electricEnergyStorage) as individual bits within the bit string. Let's assign them specific bit positions:

(a) `diesel`: bit 1

(b) `liquidPropaneGas`: bit 2

(c) `electricEnergyStorage`: bit 3

Thus, the bit string to represent "`diesel+liquidPropaneGas+electricEnergyStorage`" would have the first three bits set to 1. The bit string is:

$$1110\ 0000_2$$

We know that, in the BET pattern, the TAG for Bit String is "`03`". For the length of the data, we have 1 byte for the non used bits and 1 byte for the used bits. So, the total length is 2 bytes.
The non used bits, represented by the value 05. For the value of the Bit String, the binary sequence that represents the three active bits is:

$$1110\ 0000_2$$

Which value, in hexadecimal is:

$$E0_{16}$$

Then, the codification is:

$$03\ 02\ 01\ 68$$

Where:
03: represents the TAG,
02: represents the length of data,
05: represents the non utilized bits.
E0: represents the codified value of the BIT STRING

The XML document below is the XER encoding of Message 2 about "alter":

```xml
<?xml version="1.0" encoding="UTF-8"?>
<AlterData>
    <id>EF-456-GH</id>
    <safeDistance>
        <false />
    </safeDistance>
    <speed>4000<speed>
    <driveDirection>
        <forward />
    </driveDirection>
```

```
11          <lanePosition>
12              <detailedlanePosition>
13                  <transversalPosition>3</transversalPotion>
14                  <laneType>traffic</laneType>
15              </detailedlanePosition>
16          </lanePosition>
17      </AlterData>
```

Question 5. **(1 pt)** I injected two syntax errors in this document. For each error, give the line number and explain it.

Answer:
The syntax errors are:

Line 07:

`<speed>4000<speed>`

And the corrected version:

`<speed>4000</speed>`

Line 12:

`<transversalPosition>3</transversalPotion>`

And the corrected version:

`<transversalPosition>3</transversalPosition>`

Question 6. **(1 pt)** What would be the BER encoding of (a) positive integer 4000 ? (b) negative integer -120 ?

(a) Positive Integer 4000
We have that integer values are divided in two parts. The identifier byte, the length and the value. So, for the number 4000, we have:

- Identifier Byte: 02
- Length Bytes: 02, once 4000 needs two bytes to be represented
- Value Bytes: We have to transform 4000 to hexadecimal,

$$4000_{10} = 1111\ 1010\ 0000_2 = FA0_{16}$$

So, the value, in hexadecimal is 0FA0.

Then, the encoded value of integer 4000 is:

`02 02 0F A0`

(b) Negative Integer -120
For the negative values, we have to do the two's complement. So, for the number -120, we have:

$$120_{10} = 0111\ 1000_2$$

With the one's complement:

$$0111\ 1000_2 \rightarrow 1000\ 0111_2$$

With the two's complement, we sum 1, to the previous value:

$$1_2 + 1000\ 0111_2 = 1000\ 1000_2$$

Which value, in hexadecimal is:

`0x88`

- Identifier Byte: `02`
- Length Bytes: `01`, once -120 needs only one byte to be represented

- Value Bytes: As we've seen above, the value of -120 in hexadecimal is: `0x88`

Then, the encoded value of -120 is:

`02 01 88`

Question 7. (1 pt) Justify that the maximal encodable length in BER is 2^{1008}

Answer:
In BER encoding, the maximum encodable length is determined by the long form, where 126 bytes can be used to represent the length. Since each byte has 8 bits, this gives a total of 1008 bits available for encoding the length. The maximum value that can be encoded with 1008 bits is 2^{1008}, which represents the largest possible encodable length in BER. This system allows for the encoding of extremely large data lengths, making it highly flexible for encoding purposes.

Question 8. (1 pt) Assume we want to set the message IA5String in Message 2 about "alter" as follows:

"Warning: Automated grass cutter ahead. Maintain safe distance. Speed reduced. Hazardous debris possible. Stay alert for sudden stops and avoid lane changes near the vehicle. Thank you for your cooperation."

This message has a total of 205 characters. Give the most significant four bytes of that message encoded using BER.

Answer:
We have that the IA5String is a primitive, so, the class identifier is 00_{16}, `0` for the P/C bit, and, for the TAG evaluation, we have 22_{10} which is 16_{16}. Since the length is too big, we're going to use the Long form. In this case, since 205 is less than 256, we need 1 byte to represent the length. The first byte in the length encoding would be 0x81 (indicating 1 additional byte for the length), and the next byte is 0xCD (which is 205 in hexadecimal).
So, the final encoded value is:

`16 81 CD 57`

Where:
16: represents the TAG,
81 CD: represents the length of data,
57: represents the fist letter "W"

Question 9. (1 pt) Justify that the identifier octet for a SEQUENCE needs to be 30_{16}.

Answer:

We have that:

- **Class:** `SEQUENCE` is a universal type, represented by the bits `00`.
- **Constructed Type:** `SEQUENCE` contains other types within it, so it's marked as constructed with the bit `1`.
- **Tag Number:** The ASN.1 tag for `SEQUENCE` is 16 in decimal, which is `10000` in binary.

Combining these parts (universal `00`, constructed `1`, tag `10000`) results in the binary `0011 0000`, which converts to 30 in hexadecimal.

Question 10. (4 pts) Determine the BER-encoding for Message 2 about "alter". Justify step by step.

Answer:

Element	Type	Length	Content
`Sequence`	30_{16}	??	–
id	–	–	–
`IA5String`	16_{16}	09_{16}	"EF-456-GH"
safeDistance	–	–	–
`Boolean`	01_{16}	01_{16}	00_{16}
speed	–	–	–
`Integer`	02_{16}	02_{16}	$0F\ A0_{16}$
driveDirection	(Enumerated)	–	–
`Enumerated`	$0A_{16}$	01_{16}	01_{16}
lanePosition	(CHOICE: *simpleLanePosition* as INTEGER)	–	–
`INTEGER`	02_{16}	01_{16}	03_{16}

So, the content of `SEQUENCE` is 24 bytes, and, the length of octet is 18_{16}.

Question 11. **(1 pt)** What can go wrong with id and message being both OPTIONAL in the definition of AlterData ? Suggest additional encoding rules involving bits 8 and 7 of the identifier octet to avoid this issue.

Answer:

When both id and message fields are marked as OPTIONAL in the AlterData sequence, there's a risk that neither is provided. If neither id nor message is present in the data, it may lead to ambiguity or incomplete data, as there would be no unique identifier or descriptive message to specify the instance of AlterData.

To address this, you could introduce encoding rules using bits 8 and 7 of the identifier octet:

- **Bit 8 as Presence Indicator for `id`:** When bit 8 is set to 1, it indicates that the `id` field is present. If bit 8 is 0, `id` is absent.
- **Bit 7 as Presence Indicator for `message`:** Similarly, bit 7 can be set to 1 to indicate that `message` is present, or 0 if `message` is absent.

Question 12. **(2 pts)** The most compact ASN.1 encoding rules are the Packed Encoding Rules (PER). This is the one commonly used in 3GPP cellular technologies such as UMTS (3G), LTE (4G), or 5G. Give (4 maximum) concrete ideas for how PER greatly improves compaction with respect to BER.

Answer:

(a) **Elimination of Length Fields for Fixed-Length Data:** In PER, fields with a fixed length don't include a length indicator, reducing the encoded size. BER, however, includes length fields for every element, even if the length is already predetermined, adding unnecessary bytes.

(b) **Removal of Tags for Ordered Sequences:** PER omits tags in cases where the field type and order are known in advance. In contrast, BER encodes a tag for every field, increasing the size of the encoding.

(c) **Optimized Integer Encoding:** PER encodes integers using only the minimum number of bits required by their range, rather than fixed-width encodings. For instance, if an integer is defined within a range of 0 to 255, PER uses just 1 byte, compared to the standard integer sizes used in BER.

(d) **Efficient Handling of Optional and Default Fields:** PER compacts optional fields into a single bit in a bit-field to denote their presence or absence and skips encoding fields set to default values entirely. BER, in contrast, encodes each field and includes optional values even when they're not used, leading to a larger overall size.