

Assignment 1 IM - HL7 Message Parsing

Gerardo Vitagliano - 2017214620

Martin Schlegel - 2017190694

1 HL7 Message

The message consists of 13 segments:

1. Message Header (MSH): Here we define the receiving and sending applications/ units, the time, the message type and the version of HL7. We use version 2.7, because it is the newest one and includes the NextOfKin-segment. The message type is defined as ORU^R01 to declare it is a unsolicited observation result.
2. Patient Data (PID): In this segment we store all patient directed data, like the name, address and given contact details.
3. NextOfKin (NK1): These segments stores the information of the responsible person for the patient. In this case here are stored values related to the patients wife.
4. Observation Result (OBR): We decided to use two different OBR- Segments to save the different kinds of data (numeric and time series). The first OBR saves the numeric data in the following OBX segments.
5. 7 Observation Segments (OBX's): These segments are storing the specific numeric values, like the height, weight of the patient in different OBX segments. To show the connection to the previous defined OBR they have an ongoing index.
6. Observation Result (OBR): The second OBR stores the time series values of the Photoplethysmography (PPG) we took during the class.
7. Observation Segments (OBX): Here are the PPG values. The different values are seperated by '^'.

2 Parser

The implemented parser takes into account the structure of the message according to the HL7 standard 2.7. Due to the task requested, the parser does no full check over HL7 2.7 syntax, but instead takes into account only the specific case of a ORU^R01 message. Moreover, it implements a logic to check the correct use of segments, without further analyzing fields and subfields. In this section the MATLAB script code will be analyzed one chunk at a time, in order to explain in detail its behaviour.

The very first lines of code have the task of importing the message in the MATLAB environment and extract the relevant information to obtain a structured data containing the message. The input file is imported with the use of `textscan`: since the segments in the HL7 message are separated by a new line, the "Delimiter" option can be used to separate different text lines. This function returns a cell structure with just one cell, containing inside an Nx1 string matrix in which every row element corresponds to one of the N segment of the original message. This matrix is stored in the temporary variable 'res'.

```

1 [fileID,msg] = fopen('message.hl7');
2 inp = textscan(fileID,'%s','Delimiter','\n');
3 res=vertcat(inp{1,1});
4 fclose(fileID);

```

Code 1: Message import

Right after the segments are imported, the next task is separating fields contained in a segment. The desired output is a NxM matrix, where M is the maximum number of fields contained in a segment received. Because M is not known in advance, since segments can have a different and not fixed number of fields, there is the need of two nested for loops. The outer one will retrieve the fields for each segment, and transpose them in order to get a row vector, while the inner one, in case the number of fields of current segment is greater than the one in the previous ones, has the task to "pad" all the previous rows to fit them in the new dimensions of the matrix. At the end of this code, the variable 'field' will be an NxM string matrix, in which the (i,j) element is the j-th field of the i-th segment.

```

1 columns=1;
2 rows=length(res);
3 for i=1:rows
4     temp = strread(res{i,1},'%s','delimiter','| '); %The output is ←
        transposed so to get every segment on a row
5     if length(temp)>columns %Columns store the maximum number of fields ←
        found
6         for j=1:i           %if a new segment has more field than the ←
            previous, the previous are padded with '\p'
7             fields(j,columns+1:length(temp))={'\p'}; %to pad, the old ←
                value of columns is used until the new one
8         end
9         columns=length(temp);
10    end

```

```

11     fields(i,1:length(temp))=temp;
12 end
13
14 end

```

Code 2: Message formatting

After the message is properly imported, the code follows checking the validity of the segments. The very first lines check that the proper header, MSH, and that the second is the Patient Identification. In fact, according to HL7 2.7 syntax, these two segments are mandatory, and in this order.

```

1 if ~strcmp(fields{1,1},'MSH')
2     disp('First segment is not MSH - invalid syntax');
3     return
4 end
5 if ~strcmp(fields{2,1},'PID')
6     disp('Second segment is not Patient ID - invalid syntax');
7     return
8 end

```

Code 3: Validity of MSH and PID segments

The following check is to be sure the message sent actually contains an OBR segment, since being a Observation Request this segment is mandatory. However, given the possible flexibility in the message structure, there is no fixed position for an OBR segment: because of this a for loop is used to check all the segments. This loop starts from 3 since the two first fields, for the message to be valid, must be MSH and PID. In case at least one OBR segment is found, the variable validOBR is set to be true. Since every OBR can one or more associated OBX segments, after finding one OBR a set of check is performed in order to validate the presence and the correct ordering of the IDs for eventual OBX segments. The variable validOBX is at first set as true, since an OBX segment could not be present at all without this being an error, while is set to false in case an OBX segment is found with an invalid ID. Additional checks are present in order to consider situations in which the message ends with an OBR segment, and thus preventing an out-of-bound exception while trying to access a row index in overflow for the 'fields' matrix.

```

1 There should be at least one OBR segment, if there are any
2 validOBR=false;
3 validOBX=true;
4 for i=3:rows

```

```

5     if strcmp(fields{i,1},'OBR')
6         validOBR=true;
7         if (i+1<rows) %check if the message does ↵
            not end with an OBR segment
8             if strcmp(fields{i+1,1},'OBX')
9                 if(1~=str2num(fields{i+1,2})) %If the first OBX is not ↵
                    with ID 1
10                    validOBX=false;
11                    break
12                end
13                j=2;
14                if (i+j<rows) %Check if the message ↵
                    ends with an OBX
15                    while strcmp(fields{i+j,1},'OBX') %check if ↵
                            subsequent OBX have correct ID
16                        if (j~=str2num(fields{i+j,2}))
17                            validOBX=false;
18                            break
19                        end
20                        j=j+1;
21                    end
22                end
23            end
24        end
25    end
26 end
27
28 if ~validOBR
29     disp('There is not an OBR segment – invalid syntax');
30     return
31 end
32
33 if ~validOBX
34     disp('There is a problem with OBX segment indexes – invalid syntax');
35     return
36 end

```

Code 4: Validity of OBR/OBX segments

Finally, given the correctness of segment syntax, the value of the fields is retrieved for an output to the user. This is done using the `strsplit` MATLAB function taking into account that values in fields are separated by the `''` character. Given the formatting of the field matrix, the needed information is in fixed position: in fact even though some fields are not presents, these will result in empty cells of the matrix. This formatting was chosen rightly in order to guarantee ease of value retrieval.

```

1 name=strsplit(fields{2,6},'^');
2 height=fields{5,6};
3 heightMeasure=strsplit(fields{5,7},'^');
4
5 weight=fields{6,6};
6 weightMeasure=strsplit(fields{6,7},'^');
7
8 heartRate=fields{7,6};
9 heartMeasure=strsplit(fields{7,7},'^');
10
11 sysPre=fields{8,6};
12 sysMeasure=strsplit(fields{8,7},'^');
13
14 dyaPre=fields{9,6};
15 dyaMeasure=strsplit(fields{9,7},'^');

```

Code 5: Information Retrieval

The last lines of code simply perform output of the information retrieved from the message.

```

1 fprintf('Information about the patient: %s %s\n' ,name{1},name{2});
2 fprintf('Details: \n');
3 fprintf('Weight: %s %s\n' ,weight,weightMeasure{2});
4 fprintf('Height: %s %s\n' ,height,heightMeasure{2});
5 fprintf('Systolic blood pressure: %s %s\n' ,sysPre,sysMeasure{2});
6 fprintf('Diastolic blood pressure: %s %s\n' ,dyaPre,dyaMeasure{2});
7 fprintf('Mean blood pressure by ppg shown in figure\n');
8
9 y=str2double(strsplit(fields{13,6},'^'));
10 x=1:1000;
11 plot(x,y);
12 title('Results of PPG Examination for patient' name);
13 xlabel('Time ticks - seconds');
14 ylabel('PPG Values');

```

Code 6: Information Retrieval

2.1 Code output

Once run, the MATLAB script output is the following:

```

1 >> parser
2     Information about the patient: Botija Anacleto
3     Details:

```

```
4      Weight: 123 kg
5      Height: 176 cm
6      Systolic blood pressure: 120 mmHg
7      Dyastolic blood pressure: 88 mmHg
8      Mean blood pressure by ppg shown in figure
```

Code 7: Parser Output

