Dictionary Structure Encoding

Workflow

Momunaliev K, Ten J. Israilova N.

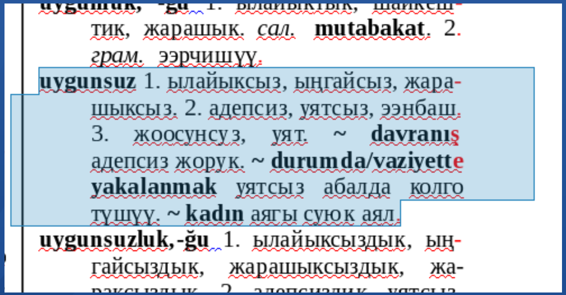
Contents

1. PDF-to-DOC CONVERSION

- We used Adobe Acrobat XI Pro‘s OCR engine,
- Save As/Export .doc option

2.PREPROCESSING

- Lines are being lost after direct doc-to-html conversion. To preserve lines information we markup each end of line in MSWord by red color. VBA macros is used. This explicit markup passes through converter.



uygunluk, -ğu 1. ылайыктык, шайкеш-
тик, жарашык. *сал.* **mutabakat**. 2.
грам. ээрчишүү.
uygunsuz 1. ылайыксыз, ыңгайсыз, жара-
шыксыз. 2. адепсиз, уятсыз, ээнбаш.
3. жоосунсуз, уят. ~ **davranış**
адепсиз жорук. ~ **durumda/vaziyette**
yakalanmak уятсыз абалда колго
түшүү. ~ **kadın** аягы суюк аял.
uygunsuzluk,-ğu 1. ылайыксыздык, ың-
гайсыздык, жарашыксыздык, жа-
раксыздык. 2. адепсиздик, уятсыз

```
Do While Not is_eod2(Selection.Range)
    Selection.EndKey Unit:=wdLine
    Selection.MoveLeft Unit:=wdCharacter, Count:=1, Extend:=wdExtend
    Selection.Font.Color = wdColorRed
    Selection.EndKey Unit:=wdLine
    Selection.MoveRight Unit:=wdCharacter, Count:=2

Loop
```

3. DOC-to-HTML CONVERSION

- After normalization html version of the source text preserves all necessary information to perform structure recognition, (see Struct1). Text information is encoded by means of CSS and HTML. Appropriate xml/parsers (e. g. ruby/nokogiri + ruby/nokogiristyles) allows us to access and control all text features. Additionally xpath expressions are more than helpful.

Struct1. Initial raw structure of html file

```
html
   body
      div+
      p+
         (b|span|i)+
                    |span*
                       |span*
        .    0 or more
        .    Occurrences
        .     of (b|span|i).
        .     Similar nodes adjacency.
```

Note: All elements under div may contain text nodes

4.DICTIONARY DATA DERIVATION

- The task is to capture only text related to dictionary entries, i.e. not page headers, footers etc. In our case every 4th div contains it. So after markup structure gets a view shown in Struct2. Additionally two columns must be delimited on the source code level. This information is required for further entries derivation.

(Struct2.) Derived Dictionary Data Structure

```
html
    body
        div[@class='DictData']+<!--page body->
        div [@class='Column_1']
            same as in (Struct1.) p pattern
        div [@class='Column_2']
            same as in (Struct1.) p pattern
```

5.NORMALIZATION

- Normalization is required to simplify further processing tasks. It eliminates adjacency of similar type elements and redundant element nesting. Structure after normalization is shown in Struct3.

```
Struct3. Normalized Initial Structure
html
    body
       div[@class='DictData']+<!--page body->
          div [@class='Column_1']
                    p+
                      (b|span|i)
                    .     0 or more
                    .     Occurrences
                    .      of (b|span|i).
                    .      No similar adjacency.

            div [@class='Column_2']
                    p+
                      (b|span|i)
                    .     0 or more
                    .     Occurrences
                    .      of (b|span|i).
                    .      No similar adjacency.
```

Note_1: No adjacent elements of the same type is allowed inside p element
Note_2: Only b, span,and i elements may contain text node

6.ENTRIES RECOGNITION AND MARKUP

- Dictionary Entries Tokens Classification is made before and after Anomalies Fix.
- Resulting structure is shown in Struct4. Tokens classifications before and after fix are shown in Fig_1 and Fig_2 respectively.
- Tokens are all <p/> elements, i. e. paragraphs

Fig_1 Tokens Classification Before Anomalies Fix

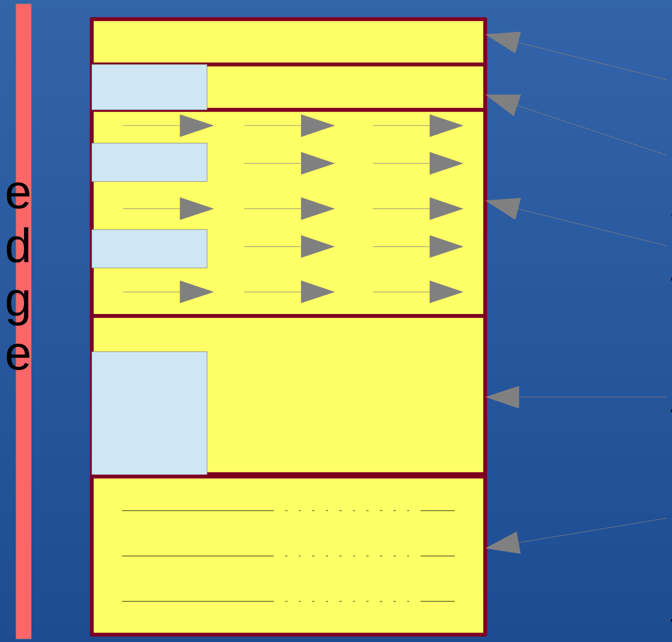


Fig_1 Descriptions:

(1) Normal case. Single line token without first line indent. Token is close to the edge.
(2) Normal case. Single or multiline token staying far from the edge
(3) Anomal case. Token has right aligned text. This kind of token may contain 1 or more normal tokens after fix
(4) Normal case. Single or multiline token with first line indent which makes it stay close to the edge.
(5) Anomal case. Multiline token without first line indent and locating close to the edge. This kind of token may contain 1 or more normal tokens after fix
Note: Anomaly fix must ensure token doesn't contain more than one entry

6.ENTRIES RECOGNITION AND MARKUP

(Struct4.) Tokens Classified After Anomaly Fix

```
html
    body
       div[@class='DictData']
          div [@class='Column_1']
                p [@class =' EntryBeginning' |
                            'Entry Continuation']
                     (b|span|i)
                     .     0 or more
                     .     Occurrences
                     .      of (b|span|i).
                     .      No similar adjacency.
          div [@class='Column_2']
                p [@class =' EntryBeginning' |
                            'Entry Continuation']
                    (b|span|i)
                     .     0 or More
                     .     Occurrences
                     .      of (b|span|i).
                     .      No similar adjacency.
```

Fig_2 Tokens Classification
After Anomalies Fix

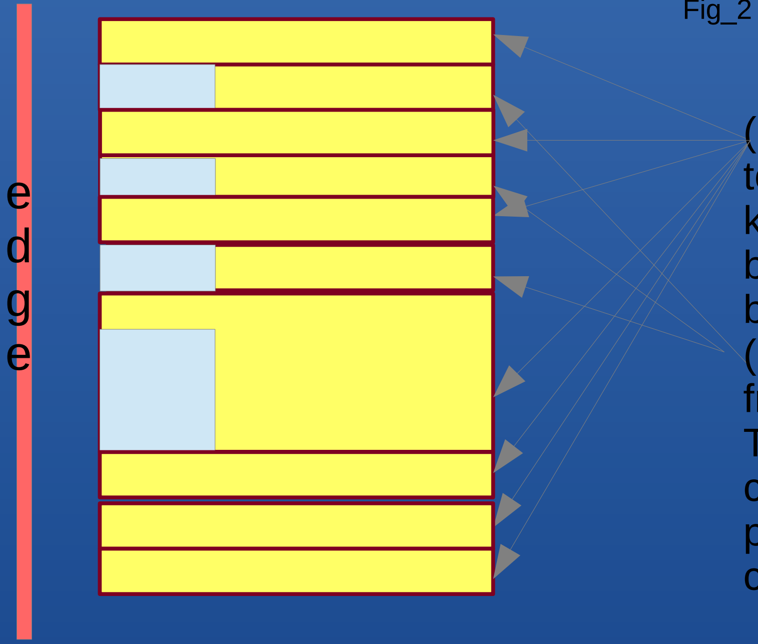


Fig_2 Descriptions:

(1) Token is close to the edge. This kind of token can be treated as entry beginning.
(2) Token is far from the edge. This kind of token can be treated as previous entry continuation .

6.ENTRIES RECOGNITION AND MARKUP

Struct5. Entries Derived Structure

```
html
    body
       div[@class='DictData']
          p [@class ='Entry']+
                b
                 (b|span|i)
                  .     Zero or More
                  .     Occurrences
                  .      of (b|span|i).
                  .      No similar adjacency.
```

Fig_3 Tokens Merged into Entries

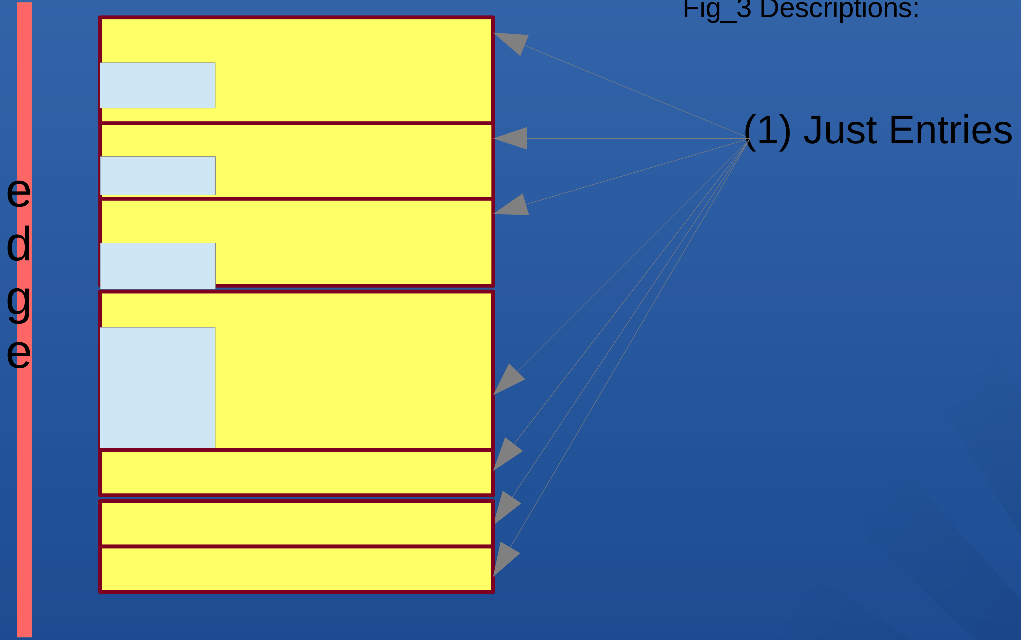


Fig_3 Descriptions:

7.FORM AND DEFINITION BLOCKS PARSING AND MARKUP

- Entry element must always start with <b> element which directly corresponds to form block. And the remainder part is left to be a sense block. Using content delimiters of these two blocks we can split them for even smaller parts such as 'orth', 'inflection' , and 'sense' sub-blocks.
- Form Block Delimiters: ':', ',', '-'.
- Sense Block Delimiters: '1.', '2.', ... 'n'.
- Resulting structure is shown in Struct6.

(Struct6.) Form and Sense Blocks and Their Top Level Components

```
html
    body
       div[@class='DictData']
        p [@class ='Entry']+
             div [@class='FormBlock']
                div[@class='orth']
                     b
                div[@class='inflection']
                     b
             div [@class='SenseBlock']
                div[@class='sense1...n']
```

Note: Leaf elements under p contain text nodes

8.SENSE ELEMENTS CONSTITUANTS

- Every sense element may contain translation equivalents delimited by ','. Additionally it may contain an example element which can be detected by <b> element as it's start and nearest from the left full stop as the end. One more subconstruction sense element may contain is reference element. This one starts by predefined abbreviation indicator and ends with nearest following full stop. Resuming the top-level structure of sense element we can draw Struct7.

Struct_7. Sense Element Top Level Components

```
html
    body
       div[@class='DictData']
          p [@class ='Entry']+
               div [@class='FormBlock']
                    div[@class='orth']
                         b
                    div[@class='inflection']
                         b
               div [@class='SenseBlock']
                    div[@class='sense_1..n']+
                        div[@class='equiv_1..k']+
                        div[@class='example ']?
                        div[@class='ref ']?
```

Thank You!

Contact:
kadyr.momunaliev@gmail.com