LexUnits and Keys

Bernard Bou, bbou at ac-toulouse.fr November 2021

Notation

Pronunciation notation in IPA requires fonts that support IPA.

See notation at the bottom, for the symbols used.

Lex Unit

Group of senses

Lexical units are defined as the **basic container of senses**. They **group senses in an ordered set**.

lexunit = [s1,..]

Where the XML version of OEWN uses *LexicalEntry* it is more appropriate to use the term *unit* here (because it is nearer to *group*, or *set*, or *block*) and use the term *entry* for the first-level member of the tuple key which is indeed an entry point (q1 below). That follows the use that FrameNet makes of the term *lexunit*: lexunits are containers that group semantic roles, their realisations and references to semantic frames.

Key

Lexical units group senses **according to a key**:

k -> lexunit

k -> [s1,..]

k -> [s+]

A key **uniquely** determines a group of senses (if the data conform to this key):

k1 -> [sa1,..]

k2 -> [sb1,..]

If we define **a sense as the pair of a key with a synset**

s = (k,S)

and this pair represents the membership bifunction

memberOf(k,S)

or

k ∈ S

then a lexunit or group of senses uniquely maps to a key (so the relation is bijective, a one-one mapping)

k1 <- [s1,..]

k1 <-> [s1,..]

A case in point is :

```
(Baroque,a) -> lexunit1 = [s1] = [baroque%3:01:00::]
(baroque,a) -> lexunit2 = [s2] = [baroque%3:01:01::]
```

each a singleton group of senses.

Now if we take it that a sense is the pair of a key with a synset,

```
s1 = baroque%3:01:00:: = (baroque,02985568-a)
s2 = baroque%3:01:01:: = (Baroque,02985568-a)
```

meaning

```
 baroque%3:01:00:: ∈ 02985568-a
 baroque%3:01:01:: ∈ 02985568-a
```

then

```
(Baroque,a) -> [s1] = [baroque%3:01:00::] = [(baroque,02985568-a)]
(baroque,a) -> [s2] = [baroque%3:01:01::] = [(Baroque,02985568-a)]
```

In this context there are two senses, with different IDs, that map to the same synset (they are both members of it):

```
baroque%3:01:00:: -> 02985568-a
baroque%3:01:01:: -> 02985568-a
```

But if we understand *sense* to mean **the concept expressed by a synset** then the relation between the key and the synset is no longer one-one.

```
(Baroque,a) -> 02985568-a = "of or relating to or characteristic of the elaborately ornamented style
 of architecture and music ..."
(baroque,a) -> 02985568-a = "of or relating to or characteristic of the elaborately ornamented style
 of architecture and music ..."
```

Tuple key

Now a key is usually a **tuple**:

k = (q1,...) = (q+)

So the lay out is:

```
q1a:
  q2a:
    q3a:
        [sa1,..]
    q3b:
        [sb1,..]
    ...
  q2b:
    q3c:
        [sc1,..]
    q3d:
        [sd1,..]
    ...
  ...
...
```

Some groupings, hence some keys, are more or less natural like *(lemma, part-of-speech)*, others arbitrary.

OEWN

This is the layout to be found in the YAML version of OEWN:

```
row:
  n-1:
    pronunciation:
    - value: ɹəʊ
    sense:
    - id: 'row%1:14:00::'
      synset: 08448447-n
    - id: 'row%1:17:00::'
      synset: 09440243-n
    - id: 'row%1:06:00::'
      synset: 03124680-n
    - id: 'row%1:14:01::'
      synset: 08450457-n
    - id: 'row%1:07:00::'
      synset: 05052831-n
      id: 'row%1:04:00::'
      synset: 00446336-n
  n-2:
    pronunciation:
    - value: ɹaʊ
    sense:
    - id: 'row%1:10:00::'
      synset: 07198809-n
  v:
    pronunciation:
    - value: ɹəʊ
    sense:
     id: 'row%2:38:00::'
     synset: 01950855-v
```

This can be simplified to:

```
row:
  n-1:
    pronunciation ɹəʊ
    lexunit 1
  n-2:
    pronunciation ɹaʊ
    lexunit 2
  v:
    pronunciation ɹəʊ
    lexunit 3
```

The containers (and so the lexical units) appear at level of indentation 3 where ordered lists of senses are to be found. There are 3 *(q1,q2) -> [s+]* mappings:

```
(row,n-1) -> [row%1:14:00::,row%1:17:00::,row%1:06:00::,row%1:14:01::,row%1:07:00::,row%1:04:00::] =
lexunit1
```

```
(row,n-2) -> [row%1:10:00::] = lexunit2
(row,v) -> [row%2:38:00::] = lexunit3
```

or

```
(row,n-1) -> lexunit1
(row,n-2) -> lexunit2
(row,v) -> lexunit3
```

or

```
(row,n,1) -> lexunit1
(row,n,2) -> lexunit2
(row,v,{}) -> lexunit3
```

where the third element of the triple is a discriminant whose possible values are currently in {1,2,{}} range across OEWN. It turns out the discriminant currently distinguishes/fleshes out different pronunciations of the lemmas. So this is equivalent to:

```
(row,n,{/ɹəʊ/}) -> lexunit1
(row,n,{/ɹaʊ/}) -> lexunit2
(row,v,{}) -> lexunit3
```

so the scheme could be a 4-tier layout:

```
row:
  n:
    {/ɹəʊ/}:
        lexunit 1
    {/ɹaʊ/}:
        lexunit 2
  v:
    {}:
        lexunit 3
```

Note that the pronunciation of the verb, although present, does not discriminate among verb lexunits, so it's irrelevant as a discriminant key but could be present.

Which leads us to define the key in OEWN (current version) as:

k=(w,t,{v*})

where w is the case-sensitive lemma, t the synset type, and {v*} the set of pronunciations

In the case of *critical* this leads us to two separate lexunits:

```
(critical,a) -> [s1,s2,s3,s5,s6]
(critical,s) -> [s4,s7]
```

where

```
s1 = oewn-critical%3:00:01:: synset=oewn-00650564-a def=marked by a tendency to find and call attenti
on to errors
s2 = oewn-critical%3:00:04:: synset=oewn-00654957-a def=at or of a point at which a property or pheno
```

```
menon suffers an abrupt change
s3 = oewn-critical%3:00:02:: synset=oewn-00652608-a def=characterized by careful evaluation
s4 = oewn-critical%5:00:00:indispensable:00 synset=oewn-00907116-s def=urgently needed
s5 = oewn-critical%3:00:03:: synset=oewn-00653599-a def=forming or having the nature of a turning point
s6 = oewn-critical%3:01:00:: synset=oewn-02841530-a def=being in or verging on a state of crisis
s7 = oewn-critical%5:00:00:crucial:00 synset=oewn-00659155-s def=of or involving or characteristic of critics
```

Other keys

Even if we leave aside pronunciation, the Princeton WordNet PWN key is different from the OEWN key:

k=(w,p)

in that the part-of-speech *p* takes part in grouping the senses, not the synset type *t*:

```
(critical,as) -> [s1,s2,s3,s4,s5,s6,s7]
```

PWN31 *index.sense* registers 7 senses for adjective *critical* (regardless of whether it's a satellite or a head):

```
critical a ... 6 00650564 00654957 00652608 00907116 00659155 00653599 02841530
```

that *index.adj* names and orders (first figure) by decreasing tag count (last figure):

```
critical%3:00:01::                    00650564    1 7
critical%3:00:04::                    00654957    2 5
critical%3:00:02::                    00652608    3 5
critical%5:00:00:indispensable:00     00907116    4 3
critical%5:00:00:crucial:00           00659155    5 2
critical%3:00:03::                    00653599    6 1
critical%3:01:00::                    02841530    7 0
```

Now this seems natural as the grouping by part-of-speech is more natural than by synset type.

If we extend this remark to OEWN, it seems desirable for OEWN to have

k=(w,p,{v*})

instead of its current key, which leads to merging adjective lexunits of types *a* and *s* into a single sense groups. The problem is how to build the new order of senses which should not consist in appending the satellites to the heads: some intermingling is needed if we want to stick to decreasing usage frequency ordering.

Other keys

Please note that another possible key could be

k=(u,cc(u),p,{v*})

with mappings like

```
(earth,Earth,n,{}) ->  unique lexunit
(baroque,Baroque,n,{}) -> unique lexunit
```

Some other examples:

k=(uc(w),p)

with mappings like (in the French Robert dictionary)

```
(ÉTÉ,n) -> unique lexunit
(CAFÉ,n) -> unique lexunit
```

k=(u,cc(u),p)

k=(w,lc(w),p)

with mappings like

```
(earth,Earth,n) -> ...
(Earth,earth,n) -> ...
```

k=(w,rd(w),p)

k=(ad(a),a,p)

k=(a,ad(a),p)

with mappings like

```
(café,cafe,n) -> unique lexunit
(café,cafe,n) -> unique lexunit
(cafe,café,n) -> unique lexunit
```

k=(w,us(w),p)

k=(w,gb(w),p)

with mappings like

```
(colour,color,n) -> unique lexunit
(color,colour,n) -> unique lexunit
```

k=(w,sz(w),p)

with mappings like

```
(realise,realize,v) -> unique lexunit
```

k=(w,sz(w),p)

with mappings like

```
(realise,realize,v) -> unique lexunit
```

Key classes

Keys have been developped in software that, when applied to data D, yield the lexunits.

k.apply(D) = {lexunit+} = {[s1,...],..} = {[s1,...]+}

In relation to a given set of data, they can have **single-value** output (the first two keys below, with OEWN) or **multi-value** results, in which case some lexunits in the data would have to be merged if the data were to conform to the key. If a key produces the empty set, it fails.

k.apply(D) = lexunit, D conforms to k

k.apply(D) = {lexunit+}, D does not conform to k

Key.OEWN is the current OEWN implementation and has the intended handling of case and pronunciations. It is a deep key in that it handles the pronunciation sets as part of the key.

```
new Key.OEWN("row", 'n', Pronunciation.ipa("ɹəʊ" ))
```

yields a unique result when applied to the data.

Key.Shallow is the current OEWN implementation and has the intended handling of case and pronunciations but, unlike the previous key, operates with a discriminant, not the set pronunciations.

```
Key.Shallow("row", 'n', "-1")
```

also yields a unique result when applied to the data.

Key.Pos is like Key.OEWN but the second element is the part-of-speech, not the synset type.

```
Key.Pos("critical", 'a', {})
```

yields multiple results when applied to the current OEWN data. OEWN's next move is now to merge the *a* and *s* lexunits so that they produce a single output. The next OEWN will conform to this key.

Key.IC ignores case in the lemma and with the current OEWN data, it is a multivalue key

```
new Key.IC("mobile", 'n', Pronunciation.ipa("ˈməʊbaɪl", "GB"), Pronunciation.ipa("ˈmoʊbil", "US"))
new Key.IC("Mobile", 'n', Pronunciation.ipa("ˈməʊbaɪl", "GB"), Pronunciation.ipa("ˈmoʊbil", "US"))
new Key.IC("MOBILE", 'n', Pronunciation.ipa("ˈməʊbaɪl", "GB"), Pronunciation.ipa("ˈmoʊbil", "US"))
```

all yield the same 2 values.

Key.PWN uses part-of-speech the way Princeton WordNet does, rather than synset type and with the current OEWN data it is a multivalue key

```
new Key.Pos("critical", 'a', Pronunciation.ipa("ˈkɹɪtɪkəl"))
```

Coding and testing

The above keys have been implemented in code to be used with the current data and the following:

```
new Key.OEWN("row", 'n', Pronunciation.ipa("ɹəʊ" ))                        .apply(model.lexes);
new Key.OEWN("critical", 'a', Pronunciation.ipa("ˈkɹɪtɪkəl"))                       .apply(model.lexe
s);

new Key.IC("mobile", 'n', Pronunciation.ipa("ˈmoʊbil", "US"), Pronunciation.ipa("ˈməʊbaɪl", "GB"))  .
apply(model.lexes);
new Key.IC("Mobile", 'n', Pronunciation.ipa("ˈməʊbaɪl", "GB"), Pronunciation.ipa("ˈmoʊbil", "US"))  .
apply(model.lexes);
```

```
new Key.IC("MOBILE", 'n', Pronunciation.ipa("ˈməʊbaɪl", "GB"), Pronunciation.ipa("ˈmoʊbil", "US"))  .
apply(model.lexes);

new Key.Shallow("row", 'n', "-1")                                       .apply(model.lexes);
new Key.Shallow("critical", 's', null)                                          .apply(model.lexes);

new Key.Pos("critical", 'a', Pronunciation.ipa("ˈkɹɪtɪkəl"))
   .apply(model.lexes);

new Key.PWN("critical", 'a')                                            .apply(model.lexes);
```

yields:

```
KEY IGNORE CASE (mobile,n,[[GB] /'məʊbaɪl/, [US] /'moʊbil/])
Mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'Mobile' mobile%1:17:00:: n 09379536-n,[1] of 'Mobile
' mobile%1:15:00:: n 09076943-n}
mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'mobile' mobile%1:06:00:: n 03781824-n}
KEY IGNORE CASE (mobile,n,[[GB] /'məʊbaɪl/, [US] /'moʊbil/])
Mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'Mobile' mobile%1:17:00:: n 09379536-n,[1] of 'Mobile
' mobile%1:15:00:: n 09076943-n}
mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'mobile' mobile%1:06:00:: n 03781824-n}
KEY IGNORE CASE (Mobile,n,[[GB] /'məʊbaɪl/, [US] /'moʊbil/])
Mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'Mobile' mobile%1:17:00:: n 09379536-n,[1] of 'Mobile
' mobile%1:15:00:: n 09076943-n}
mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'mobile' mobile%1:06:00:: n 03781824-n}
KEY IGNORE CASE (Mobile,n,[[GB] /'məʊbaɪl/, [US] /'moʊbil/])
Mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'Mobile' mobile%1:17:00:: n 09379536-n,[1] of 'Mobile
' mobile%1:15:00:: n 09076943-n}
mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'mobile' mobile%1:06:00:: n 03781824-n}
KEY IGNORE CASE (MOBILE,n,[[GB] /'məʊbaɪl/, [US] /'moʊbil/])
Mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'Mobile' mobile%1:17:00:: n 09379536-n,[1] of 'Mobile
' mobile%1:15:00:: n 09076943-n}
mobile n [GB] /'məʊbaɪl/,[US] /'moʊbil/ {[0] of 'mobile' mobile%1:06:00:: n 03781824-n}

KEY IGNORE CASE (baroque,a,[[GB] /bə'ɹɒk/, [US] /bə'ɹoʊk/])
Baroque a [GB] /bə'ɹɒk/,[US] /bə'ɹoʊk/ {[0] of 'Baroque' baroque%3:01:00:: a 02985568-a}
baroque a [GB] /bə'ɹɒk/,[US] /bə'ɹoʊk/ {[0] of 'baroque' baroque%3:01:01:: a 02985568-a}
KEY IGNORE CASE (baroque,s,[[GB] /bə'ɹɒk/, [US] /bə'ɹoʊk/])
baroque s [GB] /bə'ɹɒk/,[US] /bə'ɹoʊk/ {[0] of 'baroque' baroque%5:00:00:fancy:00 s 01799504-s}
KEY IGNORE CASE (baroque,a,[[US] /bə'ɹoʊk/, [GB] /bə'ɹɒk/])
Baroque a [GB] /bə'ɹɒk/,[US] /bə'ɹoʊk/ {[0] of 'Baroque' baroque%3:01:00:: a 02985568-a}
baroque a [GB] /bə'ɹɒk/,[US] /bə'ɹoʊk/ {[0] of 'baroque' baroque%3:01:01:: a 02985568-a}
KEY IGNORE CASE (Baroque,a,[[US] /bə'ɹoʊk/, [GB] /bə'ɹɒk/])
Baroque a [GB] /bə'ɹɒk/,[US] /bə'ɹoʊk/ {[0] of 'Baroque' baroque%3:01:00:: a 02985568-a}
baroque a [GB] /bə'ɹɒk/,[US] /bə'ɹoʊk/ {[0] of 'baroque' baroque%3:01:01:: a 02985568-a}

KEY (bass,n,[/beɪs/])
bass n-1 /beɪs/ {[0] of 'bass' bass%1:07:01:: n 04994045-n,[1] of 'bass' bass%1:10:01:: n 07045779-n,
[2] of 'bass' bass%1:18:00:: n 09861916-n,[3] of 'bass' bass%1:10:00:: n 06885404-n,[4] of 'bass' bas
s%1:06:02:: n 02806515-n}
KEY SHALLOW (bass,n,-1)
bass n-1 /beɪs/ {[0] of 'bass' bass%1:07:01:: n 04994045-n,[1] of 'bass' bass%1:10:01:: n 07045779-n,
[2] of 'bass' bass%1:18:00:: n 09861916-n,[3] of 'bass' bass%1:10:00:: n 06885404-n,[4] of 'bass' bas
s%1:06:02:: n 02806515-n}
KEY (bass,n,[/bæs/])
bass n-2 /bæs/ {[0] of 'bass' bass%1:13:02:: n 07793921-n,[1] of 'bass' bass%1:13:01:: n 07793488-n,[
2] of 'bass' bass%1:05:00:: n 02568204-n}
KEY SHALLOW (bass,n,-2)
bass n-2 /bæs/ {[0] of 'bass' bass%1:13:02:: n 07793921-n,[1] of 'bass' bass%1:13:01:: n 07793488-n,[
2] of 'bass' bass%1:05:00:: n 02568204-n}
```

```
KEY IGNORE CASE (Earth,n,[[GB] /ɜːθ/, [US] /ɝθ/])
Earth n [GB] /ɜːθ/,[US] /ɝθ/ {[0] of 'Earth' earth%1:17:00:: n 09293800-n,[1] of 'Earth' earth%1:15:00:: n 08579604-n}
earth n [GB] /ɜːθ/,[US] /ɝθ/ {[0] of 'earth' earth%1:17:02:: n 09293800-n,[1] of 'earth' earth%1:27:00:: n 14867162-n,[2] of 'earth' earth%1:17:01:: n 09357302-n,[3] of 'earth' earth%1:15:01:: n 08579604-n,[4] of 'earth' earth%1:27:01:: n 14868584-n,[5] of 'earth' earth%1:09:00:: n 05678816-n,[6] of 'earth' earth%1:06:00:: n 03467679-n}
KEY IGNORE CASE (Earth,n,[[GB] /ɜːθ/, [US] /ɝθ/])
Earth n [GB] /ɜːθ/,[US] /ɝθ/ {[0] of 'Earth' earth%1:17:00:: n 09293800-n,[1] of 'Earth' earth%1:15:00:: n 08579604-n}
earth n [GB] /ɜːθ/,[US] /ɝθ/ {[0] of 'earth' earth%1:17:02:: n 09293800-n,[1] of 'earth' earth%1:27:00:: n 14867162-n,[2] of 'earth' earth%1:17:01:: n 09357302-n,[3] of 'earth' earth%1:15:01:: n 08579604-n,[4] of 'earth' earth%1:27:01:: n 14868584-n,[5] of 'earth' earth%1:09:00:: n 05678816-n,[6] of 'earth' earth%1:06:00:: n 03467679-n}

KEY (row,n,[/ɹəʊ/])
row n-1 /ɹəʊ/ {[0] of 'row' row%1:14:00:: n 08448447-n,[1] of 'row' row%1:17:00:: n 09440243-n,[2] of 'row' row%1:06:00:: n 03124680-n,[3] of 'row' row%1:14:01:: n 08450457-n,[4] of 'row' row%1:07:00:: n 05052831-n,[5] of 'row' row%1:04:00:: n 00446336-n}
KEY SHALLOW (row,n,-1)
row n-1 /ɹəʊ/ {[0] of 'row' row%1:14:00:: n 08448447-n,[1] of 'row' row%1:17:00:: n 09440243-n,[2] of 'row' row%1:06:00:: n 03124680-n,[3] of 'row' row%1:14:01:: n 08450457-n,[4] of 'row' row%1:07:00:: n 05052831-n,[5] of 'row' row%1:04:00:: n 00446336-n}
KEY (row,n,[/ɹaʊ/])
row n-2 /ɹaʊ/ {[0] of 'row' row%1:10:00:: n 07198809-n}
KEY SHALLOW (row,n,-2)
row n-2 /ɹaʊ/ {[0] of 'row' row%1:10:00:: n 07198809-n}

KEY (critical,a,[/ˈkɹɪtɪkəl/])
critical a /ˈkɹɪtɪkəl/ {[0] of 'critical' critical%3:00:01:: a 00650564-a,[1] of 'critical' critical%3:00:04:: a 00654957-a,[2] of 'critical' critical%3:00:02:: a 00652608-a,[3] of 'critical' critical%3:00:03:: a 00653599-a,[4] of 'critical' critical%3:01:00:: a 02841530-a}
KEY SHALLOW (critical,s,null)
critical s /ˈkɹɪtɪkəl/ {[0] of 'critical' critical%5:00:00:indispensable:00 s 00907116-s,[1] of 'critical' critical%5:00:00:crucial:00 s 00659155-s}
KEY POS (critical,a,[/ˈkɹɪtɪkəl/])
critical a /ˈkɹɪtɪkəl/ {[0] of 'critical' critical%3:00:01:: a 00650564-a,[1] of 'critical' critical%3:00:04:: a 00654957-a,[2] of 'critical' critical%3:00:02:: a 00652608-a,[3] of 'critical' critical%3:00:03:: a 00653599-a,[4] of 'critical' critical%3:01:00:: a 02841530-a}
KEY PWN (critical,a)
critical a /ˈkɹɪtɪkəl/ {[0] of 'critical' critical%3:00:01:: a 00650564-a,[1] of 'critical' critical%3:00:04:: a 00654957-a,[2] of 'critical' critical%3:00:02:: a 00652608-a,[3] of 'critical' critical%3:00:03:: a 00653599-a,[4] of 'critical' critical%3:01:00:: a 02841530-a}
critical s /ˈkɹɪtɪkəl/ {[0] of 'critical' critical%5:00:00:indispensable:00 s 00907116-s,[1] of 'critical' critical%5:00:00:crucial:00 s 00659155-s}
```

Notation

e+ stands for the repetition of at least one element of type e

e* also stands the repetition of an element of type e, possibly none

(e1,..) is a tuple

(e+) is a tuple of at least one element

(e) is an one-tuple = **e**

{e1,..} is an unordered set of at least one element

{e+} is an unordered set of at least one element = **{e1,..}**

{e*} is an unordered set, possibly empty

{} is the empty set

{e} is the singleton set containing e

[e1,..] is an ordered set of at least one element

[e+] is an ordered set of at least one element = **[e1,..]**

[e*] is an ordered set, possibly empty

f(e) is the result of applying function f to e

t:e is the element tagged by tag, it is equivalent to the pair **(e,t)**

e can be thought of as tagged by {} in some contexts, it is equivalent to the pair **(e,{})**

D stands for the set of data, OEWN or PWN

w stands for an element of type **case-sensitive written form** *play*,*Shakespeare*,*baroque*,*Baroque*,*realize*,*realise*,*color*,*colour*,*battle of Verdun*,*Battle of Verdun*,*naive*,*naïve*,*cafe*,*café* Note that capitalisation and use of diacritics make different elements.

u stands for **lower cased written form** u=lc(w)

v stands for an element of type **pronunciation** (*v* is for vocal) /ɹəʊ/,/ɹaʊ/ are different pronunciations They are possibly tagged by country. *GB: /ˈfɑːðə(ɹ)/*,*US: /ˈfɑðɚ/*,*AU: /ˈfaːðə/* The untagged case being *{}: /ˈfæðəɹ/*

t stands for synset_type with **T={n,v,a,r,s}** where *a* is a head adjective and s a satellite adjective

p stands for part-of-speech with **P={n,v,as,r}** where *as* is an adjective (heads or satellites are merged)

m stands for a morphological form lile *saw*,*seen*,*was*,*were*

a stands for forms without diacritics like *naive*,*cliche*,*cafe*,*facade*,*fete*,*neglige*,*Chloe*

d stands for forms with diacritics like *naïve*,*cliché*,*café*,*fête*,*négligé*,*Chlöe*,*Chlöé,façade*

a stands for forms without diacritics like *naive*,*cliche*,*cafe*,*fete*,*neglige*,*Chloe*,*facade*

Usual functions

lc() lower case conversion *lc(Shakespeare)=shakespeare lc(Baroque)=baroque lc(DoD)=dod*

cc() camel case conversion and suchlike *cc(shakespeare)=Shakespeare cc(william shakespeare)=William Shakespeare cc(baroque)=Baroque* Note that not necessarily the first character, not necessarily all those at word-start position are upper-cased and this transformation can yield multiple results (as such it does not seem to be very predictable): *cc(dod)=DoD cc(battle of Verdun)={battle of Verdun,Battle of Verdun} cc(ddi)={DDI,ddI} cc(ddc)={DDC,ddC}*

uc() uppercase conversion of all characters *uc(Shakespeare)=SHAKESPEARE*

rp() remove period *rp(D.C.)=DC rp(B.C.)=BC rp(Dr.)=Dr*

rd() remove diacritics *rd(fête)=fete rd(café)=cafe rd(façade)=facade rd(négligé)={neglige,negligee}*

ad() add diacritics *ad(fete)=fête ad(cafe)=café ad(facade)=façade ad(neglige)=négligé ad(negligee)=négligée*

sz() s-to-z transform *sz(realise)=realize*

zs() z-to-s transform *zs(realize)=realise*

us() americanise *us(colour)=color us(mobile)=mobile us(GB:/ˈməʊbaɪl/)=US:/ˈmoʊbil/*

gb() anglicise *gb(color)=colour gb(mobile)=mobile gb(US:/ˈmoʊbil/)=GB:/ˈməʊbaɪl/*

mf() set of morphological forms *mf(be)={was,were} mf(mouse)={mice}*

These can be combined: *lc(rp(Dr.))=dr* sometimes resulting in the identity function *sz(zs(realise))=realise us(gb(colour))=colour*