Recoding nominal data

Michael Cysouw cysouw@mac.com

A common situation in comparative linguistic data collection is that data is encoded as nominal ('categorical') attributes. An attribute is conceived as a finite list of possibilities, viz. the values of the attribute. Although this is of course a completely noremal and widespread practice in data encoding, in comparative linguistics it is mostly not trivial to decide on the delimitation of the different values. There is often ample discussion about the best way to separate the attested variation into types (terms like 'types', 'categories' or 'values' will the considered as equivalent here), and in general there is often no optimal or preferred way how to define the different types.

In practice then, different scholars will often like to interpret data differently. One common wish is to be able to recode data that has already been categorised into types by a different. Note that such a recoding of course will never be able to easily split types, because for that goal a complete reconsideration of all underlying raw data is necessary, something that will not be further considered here.

Given any given nominal data set (like for example the WALS data, as included in this package), there are various transformations that are often requested, and that are reasonably easy to perform: *merge values* and *split attributes*. A third kind recoding, *merge attributes* is also possible, but needs a bit more effort. Furthermore, the actual recoding will consist of just a few lists of information, which allows for an easy way to share and publish the actual recoding decisions.

As an example, consider the following toy data frame with two attributes `size` and `kind`.

```
data <- data.frame(
    size = c('large','very small','small','large','very small','tiny'),
    kind = c('young male','young female','old female','old male',NA,'young female'),
    row.names = paste('obs', 1:6, sep='')
    )
data
```

```
##            size         kind
## obs1      large   young male
## obs2 very small young female
## obs3      small   old female
## obs4      large     old male
## obs5 very small         <NA>
## obs6       tiny young female
```

Merge values

The first kind of recoding to be exemplified here is to merge values. The first attribute of in our data has four values: `large`, `small`, `very small` and `tiny`. Suppose we would like to merge the values `small`, `very small` and `tiny` into one value value `small`. What we have to do is to define a new attribute with new values, and link the original values to our new values. In practice such a recoding looks as shown below: a list with a new name for the new attribute (`attribute=`), names for the new values of this attribute (`values=`), the name of the original attribute that is to be recoded (`recodingOf=`), and the central informatin for the recoding: the link-vector (`link=`).

The link-vector has the same length as to the number of values of the original attribute in the order as given by `levels(data$size)` viz. in this case large, small, tiny, very small. These four original values are linked

to the new values as specified in the link-vector: the first original value is linked to the first new value (1), the second original value is linked to the second new value (2), the third original value to the second new value (2), etc. A zero in this link-vector is designated for values that should not be linked (i.e. NA, but that does not work in YAML, see below.)

The function `recode` taked the original data and the recoding-specification, and returns the new, recoded, data:

```
# Specifying a recoding
recoding <- list(
  list(
    recodingOf = 'size',
    attribute = 'newSize',
    values = c('large','small'),
    link = c(1,2,2,2)
    )
  )
# Do the actual recoding and see the results
recode(data, recoding)
```

```
##      newSize
## obs1   large
## obs2   small
## obs3   small
## obs4   large
## obs5   small
## obs6   small
```

Split attributes

The recoding-object has a doubly-embedded list structure, which might seem superfluous, but this is because the example above only specifies a single new attribute. To specify more than one attribute, simply various such specification can be listed, as illustrated below. In the following example, the second original attribute (`kind`) is split into two new attributes (`gender` and `age`), but such a split is simply represented as two different ways of merging the values. In total, our recoding example has now been extended to recoding three different new attributes.

```
# Specifying the recoding of three different new attributes
recoding <- list(
  list(
    recodingOf = 'size',
    attribute = 'size',
    values = c('large','small'),
    link = c(1,2,2,2)
    ),
  list(
    recodingOf = 'kind',
    attribute = 'gender',
    values = c('female','male'),
    link = c(1,2,1,2)
    ),
  list(
    recodingOf = 'kind',
```

```
    attribute = 'age',
    values = c('old','young'),
    link = c(1,1,2,2)
    )
  )
# Do the recoding and show it
newdata <- recode(data, recoding)
newdata
```

```
##       size gender   age
## obs1 large   male young
## obs2 small female young
## obs3 small female   old
## obs4 large   male   old
## obs5 small   <NA>  <NA>
## obs6 small female young
```

Merge attributes

Combining various attributes into a singly new attribute works very similar, only that there are multiple attributes specified at `recodingOf`. Note that there are various zeros in the link-vector, specifying that some value-combinations are not to be linked.

```
back_recoding <- list(
  list(
    recodingOf = c('size','age'),
    attribute = 'size+age',
    values = c('large+old','large+young','small+old','small+young'),
    link = c(1,3,0,2,4,0,0,0,0,0)
    )
  )
recode(newdata, back_recoding)
```

```
##         size.age
## obs1 large+young
## obs2 small+young
## obs3   small+old
## obs4   large+old
## obs5        <NA>
## obs6 small+young
```

To get the indices (including zeros) of the link-vector right, one has to realize that the recoding of two attributes is based on the cross-section of the values from the two attributes, including possible NAs. Internally, this uses the function `expand.grid`, leading in the current example to the following four values to be recoded. For larger mergers (with multiple attributes, or with attributes that have many values) this can become rather tedious, because there are very many possible combinations that all have to linked in the link-vector.

```
expand.grid(c(levels(newdata$size),NA),c(levels(newdata$age),NA))
```

```
##    Var1  Var2
## 1 large   old
```

```
## 2 small   old
## 3  <NA>   old
## 4 large young
## 5 small young
## 6  <NA> young
## 7 large  <NA>
## 8 small  <NA>
## 9  <NA>  <NA>
```

Using recoding templates

Specifying recodings is often rather tedious within R. Also, the resulting nested list datastructure in R is not very insightful to share or publish. As an alternative, I propose to use a YAML representation of the recoding for editing and sharing. The function `write.recoding.template` can be used to produce a template that can then be manually edited. All the necessary information for the recoding will be included in the file.

The list of the attributes that one wants to recode should be specified as a **list** in the function `write.recoding.template`. In that way it is possible to both recode individual attributes, but also combinations of attributes. For example, `write.recoding.template( list(1, c(1,2)), data = data, file = file)` will write the following YAML information to `file`. The tildes `~` show the missing information to be added. Note that the second recoding is a combination of two attributes.

The function `recode` also accepts a path to a YAML-file as an input of a recoding.

```
## title: ~
## author: ~
## date: '2014-06-06'
## original_data: ~
## recoding:
## - recodingOf: size
##   attribute: ~
##   values:
##   - ~
##   - ~
##   link: ~
##   originalValues:
##   - large
##   - small
##   - tiny
##   - very small
## - recodingOf:
##   - size
##   - kind
##   attribute: ~
##   values:
##   - ~
##   - ~
##   link: ~
##   originalValues:
##     '1': large + old female
##     '2': small + old female
##     '3': tiny + old female
##     '4': very small + old female
##     '5': NA + old female
```

```
##     '6': large + old male
##     '7': small + old male
##     '8': tiny + old male
##     '9': very small + old male
##     '10': NA + old male
##     '11': large + young female
##     '12': small + young female
##     '13': tiny + young female
##     '14': very small + young female
##     '15': NA + young female
##     '16': large + young male
##     '17': small + young male
##     '18': tiny + young male
##     '19': very small + young male
##     '20': NA + young male
##     '21': large + NA
##     '22': small + NA
##     '23': tiny + NA
##     '24': very small + NA
##     '25': NA + NA
```

Using recoding shortcuts

It is of course also possible to just manually write a recoding structure, either directly as a list within R or as a YAML-file. To make this even easier, the function `read.recoding` (used internally in `recode` as well) allows for various shortcuts in the formulation of a recoding:

- **Order is unimportant:** Because every recoding is a labelled list, the ordering of the specifications can be entered at will. The order will be harmonized by `read.recoding`
- **Abbreviate labels:** The labels of the specifications (like `attribute` or `link`) can be abbreviated, and in practice the first letter suffices.
- **Leave out names:** For a recoding, only `link` and `recodingOf` are minimally necessary. New attribute and value names are added automatically when nothing is specified. The automatically specified names are not very useful though (they look like `Att1` or `Val4`). Manual specification of names is strongly preferred.
- **Keep original attribute by not specifying `link`:** When no `link` is specified, the original attribute from `recodingOf` will simply be copied verbatim, without any recoding.
- **Use column numbers instead of attribute names:** Instead of the names of the original attributes it is also possible to specify the number of the column in the original data frame.
- **Use `doNotRecode` to keep original attributes:** To add original attributes without recoding them it is also possible to use `doNotRecode=` (possible abbreviated as `d=`), followed by a vector with the column numbers of the original data to be copied.

To illustrate these possibilities, consider the following recoding of our toy dataset:

```
short_recoding <- list(
  # same as first example at the start of this vignette
  # using abbreviations and a different order
  list(
    r = 'size',
    a = 'newSize',
    l = c(1,2,2,2),
```

```
    v = c('large','small')
    ),
  # same new attribute, but with automatically generated names
  list(
    r = 'size',
    l = c(1,2,2,2)
    ),
  # keep original attribute in column 2 of the data
  list(
    r = 2
    ),
  # add three times the first original attribute
  # senseless, but it illustrates the possibilities
  list(
    d = c(1,1,1)
    )
  )
recode(data, short_recoding)
```

```
##      newSize Att2         kind       size     size.1     size.2
## obs1   large val1   young male      large      large      large
## obs2   small val2 young female very small very small very small
## obs3   small val2   old female      small      small      small
## obs4   large val1     old male      large      large      large
## obs5   small val2         <NA> very small very small very small
## obs6   small val2 young female       tiny       tiny       tiny
```

Note that this short_recoding would be really short when written manually in YAML:

```
recoding:
- r: size
  a: newSize
  v: [large, small]
  l: [1,2,2,2]
- r: size
  l: [1,2,2,2]
- r: 2
- d: [1,1,1]
```

To document the recoding, it is to be preferred to expand all the shortcuts to their full text. This can be done by using `read.recoding`. When `file` is specified here, then the result is written to a YAML file that can be easily shared or published as documentation of the recoding.

```
read.recoding(short_recoding, file = yourFile , data =  data)
```

```
## title: ~
## author: ~
## date: '2014-06-06'
## recoding:
## - recodingOf: size
##   attribute: newSize
##   values:
```

```
##   - large
##   - small
##   link:
##   - 1
##   - 2
##   - 2
##   - 2
##   originalValues:
##   - large
##   - small
##   - tiny
##   - very small
## - recodingOf: size
##   attribute: Att2
##   values:
##   - val1
##   - val2
##   - val3
##   - val4
##   link:
##   - 1
##   - 2
##   - 2
##   - 2
##   originalValues:
##   - large
##   - small
##   - tiny
##   - very small
## - doNotRecode: kind
## - doNotRecode:
##   - size
##   - size
##   - size
```