Using drawProteins

Dr Paul Brennan[*1]

[1]Centre for Medical Education, School of Medicine, Cardiff University, Cardiff, Wales, United Kingdom

*BrennanP@cardiff.ac.uk

2017-08-17

Contents

1 Overview of drawProteins

This package has been created to allow the creation of protein schematics based on the data obtained from the Uniprot Protein Database.

The basic workflow is: 1. to provide one or more Uniprot IDs 2. get a list of feature from the Uniprot API 3. draw the basic chains of these proteins 4. add features as desired

drawProteins uses the package httr to interact with the Uniprot API and extract a JSON object into R. The JSON object is used to create a data.table.

The graphing package ggplot2 is then used to create the protein schematic.

The package works best when pipes %>% provided by the magrittr package are also used.

2 Getting the data from Uniprot

Currently, drawProteins interacts with the [Uniprot database]http://www.uniprot.org/. At least one working Uniprot accession numbers must be provided. More than one can be provided but they must be separated by a single space. The spaces are replaced to create an url that can be used to query the Uniprot API

The get_features() function uses the Uniprot API to return the features of a protein - the chain, domain information and other annotated features such as "repeats" and "motifs". Post-translational modifications, such as phosphorylations, are also provided.

The httr::content() function is then used to extract the content. From the get_features() function, this will provide lists of lists. The length of the parent lists corresponds to the number of accession numbers provided. Interestingly, the order sometimes appears different to that provided. Each of lists inside the parent list are a list of six - one for each protein - that contains names of the proteins and the features.

An example List of 1, which then contains a List of 6, is provided as data in the package entitled protein_json. These are the features of a protein called Rel A or NF-kappaB, p65, a well studied transcription factor.

Without internet access, this can be pulled into the environment with this code:

```
data("protein_json")
```

With internet access, this can be retreived from Uniprot with this code:

```
# accession numbers of rel A
"Q04206" %>%
  drawProteins::get_features() ->
  protein_json
```

3 Turning Uniprot data into a dataframe

The next step in the workflow is to convert the data from the Uniprot API into a dataframe that can be used with ggplot2.

The feature_to_dataframe() function will convert the list of lists of six provided by the get_features() function to a dataframe which can then be used to plot the schematics.

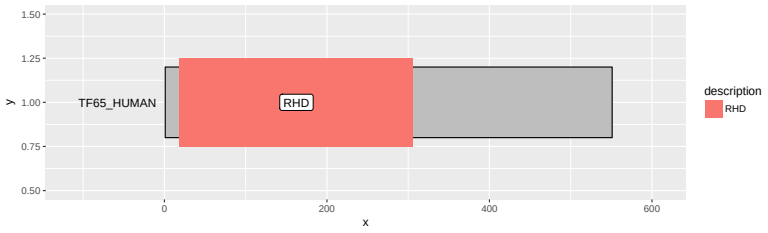
The feature_to_dataframe() function will also add an "order" value to allow plotting. The order goes from the bottom in the manner of a graph.

```
# included data in package of one protein
  protein_json %>%
  drawProteins::feature_to_dataframe() ->
  prot_data

# show in console
  head(prot_data[1:4])
                type                 description begin end
featuresTemp    CHAIN    Transcription factor p65     1 551
featuresTemp1  DOMAIN                         RHD    19 306
featuresTemp2  REGION          Activation domain   415 459
featuresTemp3   MOTIF Nuclear localization signal   301 304
featuresTemp4   MOTIF                      9aaTAD   536 544
featuresTemp5 MOD_RES        N-acetylmethionine     1   1
```

4 Plot the data

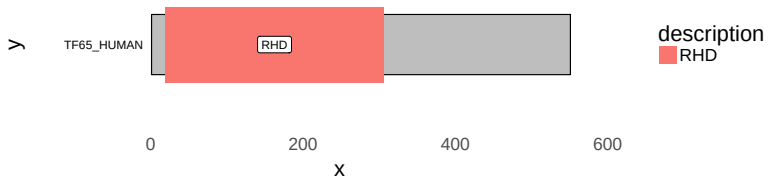
The data can be plotted with ggplot2 using the geom_rect() and geom_label. These have been combined into a geom_chain() function that can be used with pipes in the following way.

```
prot_data %>%
  geom_chains() %>%
  geom_domains -> p
p
```



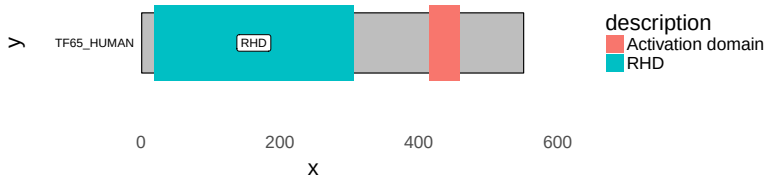
To show this visualisation better, a white background helps as well as removing the y-axis and the grid. Also changing the size of the text using the base_size argument. This can be done with this code:

```
# white background and remove y-axis
p <- p + theme_bw(base_size = 20) +  # white background
  theme(panel.grid.minor=element_blank(),
        panel.grid.major=element_blank()) +
  theme(axis.ticks = element_blank(),
        axis.text.y = element_blank()) +
  theme(panel.border = element_blank())
p
```

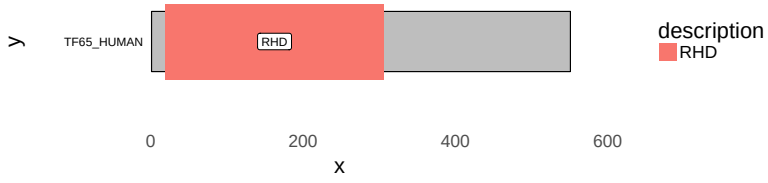


5 Checking the other features

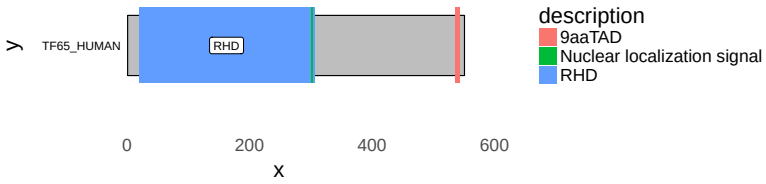
```
p %>% geom_region  # adds activation domain
```



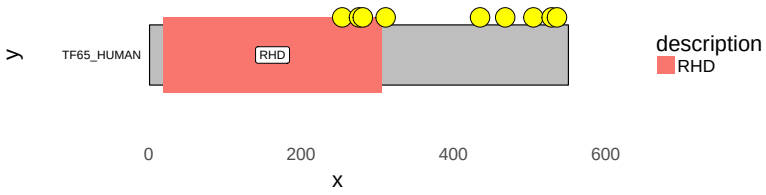

```
p %>% geom_repeat  # doesn't add anything in this case
```




```
p %>% geom_motif   # adds 9aa Transactivation domain & NLS
```




```
p %>% geom_phospho(size = 8)  # adds phosphorylation sites from Uniprot
```

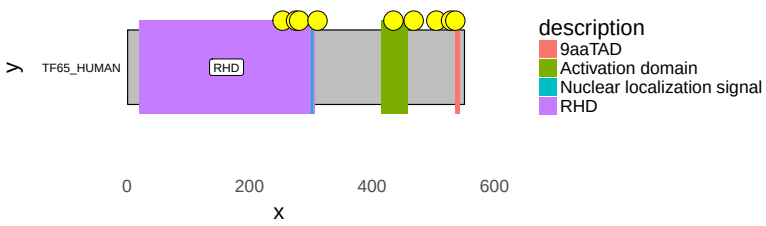


6 Putting it all together

In this way it's possible to chose the geoms that give the information desired in the way you like. Some customisation is possible as described below.

For Rel A, my recommendation would be the following workflow.

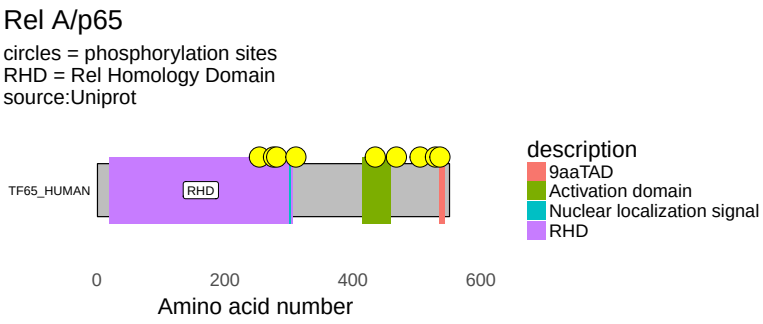
```
prot_data %>%
  geom_chains() %>%
  geom_domains() %>%
  geom_region %>%
  geom_motif %>%
  geom_phospho(size = 8) -> p

p <- p + theme_bw(base_size = 20) +  # white background and change text size
  theme(panel.grid.minor=element_blank(),
        panel.grid.major=element_blank()) +
  theme(axis.ticks = element_blank(),
        axis.text.y = element_blank()) +
  theme(panel.border = element_blank())

p
```



7 Adding titles to the plots

Using ggplot2 then allows the addition of titles:

```
# add titles
p <- p + labs(x = "Amino acid number",         # label x-axis
              y = "",  # label y-axis
              title = "Rel A/p65",
              subtitle = "circles = phosphorylation sites\nRHD = Rel Homology Domain\nsource:Uniprot")
p
```



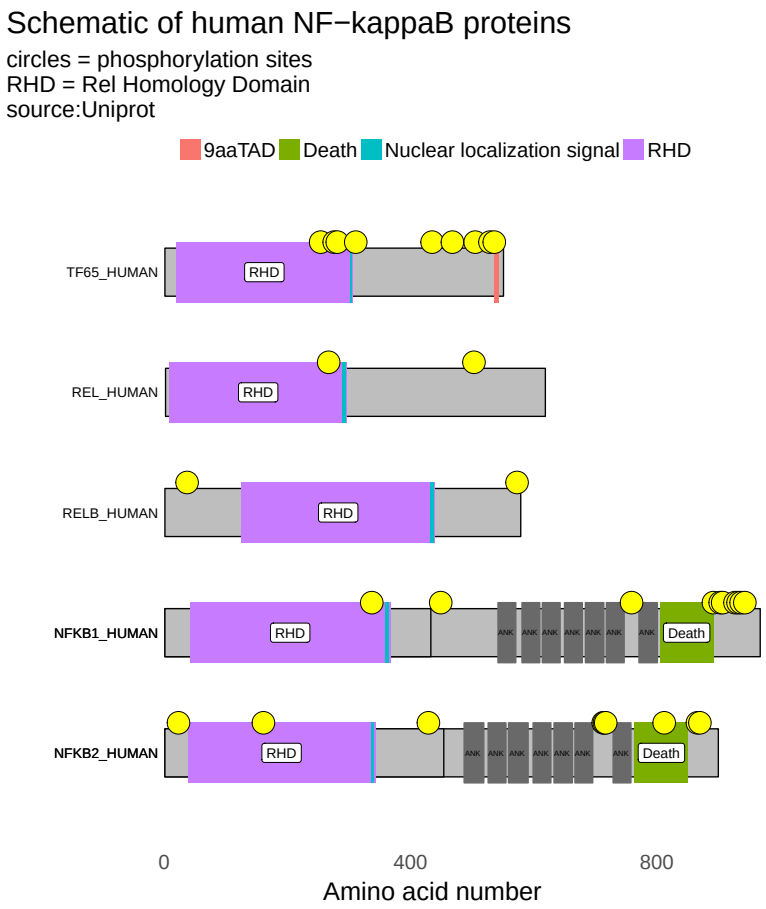
7.1 Drawing schematic for multiple proteins

With internet access, the script below shows the workflow for five proteins of the NFkappaB transcription factor family.

```
# accession numbers of five NF-kappaB proteins
"Q04206 Q01201 Q04864 P19838 Q00653" %>%
  drawProteins::get_features() %>%
  drawProteins::feature_to_dataframe() ->
  prot_data
[1] "Download has worked"

prot_data %>%
  geom_chains() %>%
  geom_domains() %>%
  geom_repeat() %>%
  geom_motif %>%
  geom_phospho(size = 8) -> p

# background and y-axis
p <- p + theme_bw(base_size = 20) +  # white background and change text size
  theme(panel.grid.minor=element_blank(),
        panel.grid.major=element_blank()) +
  theme(axis.ticks = element_blank(),
```

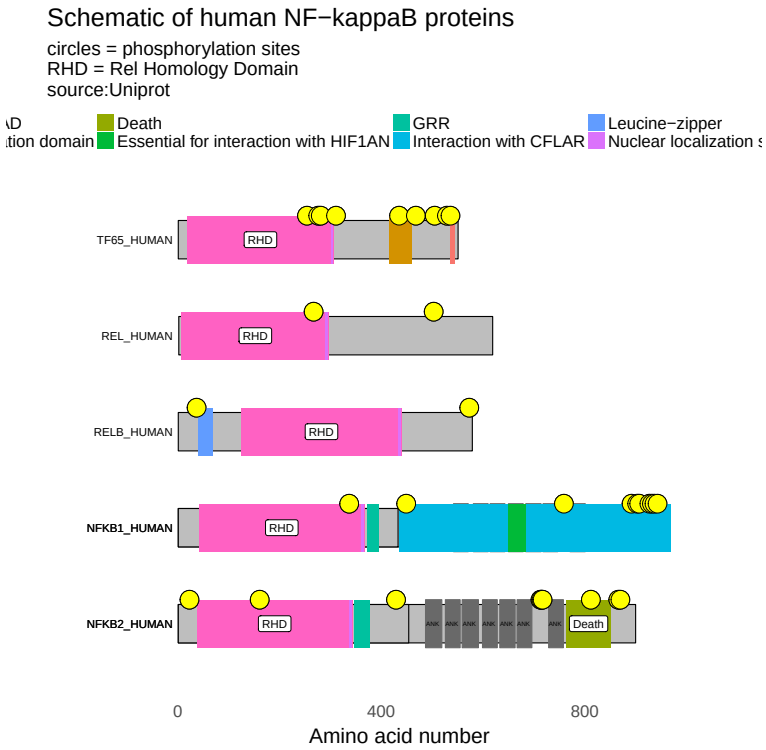
Using drawProteins

```
        axis.text.y = element_blank()) +
  theme(panel.border = element_blank())

# add titles
p <- p + labs(x = "Amino acid number",         # label x-axis
              y = "",  # label y-axis
              title = "Schematic of human NF-kappaB proteins",
              subtitle = "circles = phosphorylation sites\nRHD = Rel Homology Domain\nsource:Uniprot")


# move legend to top
p <- p + theme(legend.position="top") + labs(fill="")

p
```



Adding geom_region give lots more information but perhaps adds confusion. What do you think?

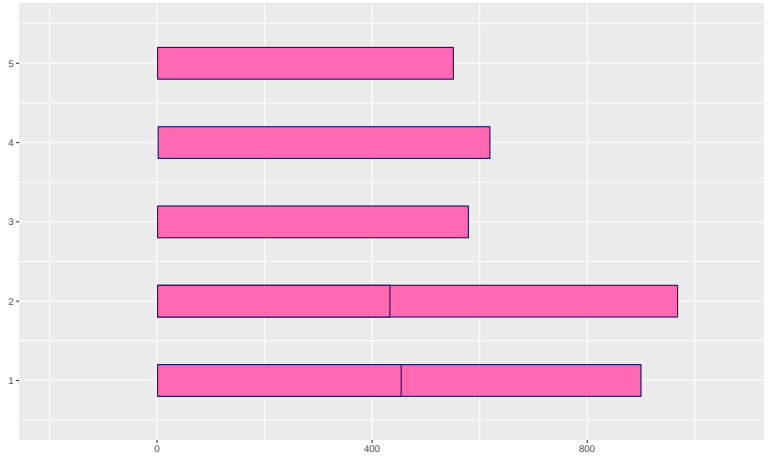
```
p %>% geom_region %>% geom_motif %>% geom_phospho(size = 8)
```



8 Customising the geoms

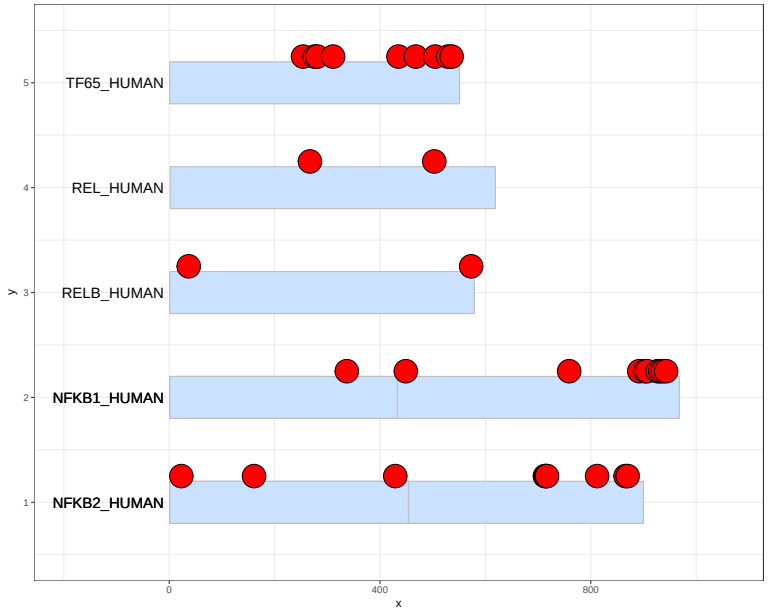
Currently, it's possible to customise the chain colour and outline. It's possible to remove the labels.

```
prot_data %>%
  geom_chains(label_chains = FALSE,
              fill = "hotpink",
              outline = "midnightblue")
```



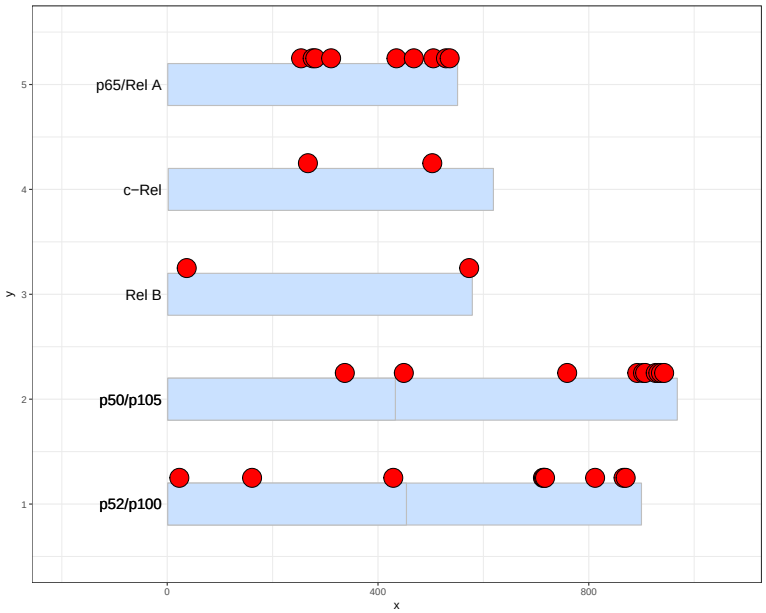
It's also possible to change the size and colour of the phosphorylation symbols.

```
prot_data %>%
  geom_chains(fill = "lightsteelblue1",
              outline = "grey",
              label_size = 5) %>%
  geom_phospho(size = 10, fill = "red") -> p

p + theme_bw()
```



It's also possible to change the labels to a custom list. But remember that the plots are drawn from the bottom up.

```
prot_data %>%
  geom_chains(fill = "lightsteelblue1",
              outline = "grey",
              labels = c("p52/p100",
                         "p52/p100",
                         "p50/p105",
                         "p50/p105",
                         "Rel B",
                         "c-Rel",
                         "p65/Rel A"),
              label_size = 5) %>%
  geom_phospho(size = 8, fill = "red") -> p

p + theme_bw()
```



9 Session info

Here is the output of sessionInfo() on the system on which this document was compiled:

```
R version 3.3.2 (2016-10-31)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: macOS Sierra 10.12.6
```

```
locale:
[1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
[7] base

other attached packages:
[1] knitr_1.17              ggplot2_2.2.1
[3] httr_1.3.0              drawProteins_0.0.0.9000
[5] magrittr_1.5            BiocStyle_2.2.1

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.12     munsell_0.4.3    colorspace_1.3-2
 [4] R6_2.2.2         rlang_0.1.2      stringr_1.2.0
 [7] plyr_1.8.4       tools_3.3.2      grid_3.3.2
[10] gtable_0.2.0     htmltools_0.3.6  yaml_2.1.14
[13] lazyeval_0.2.0   rprojroot_1.2    digest_0.6.12
[16] tibble_1.3.3     curl_2.8.1       evaluate_0.10.1
[19] rmarkdown_1.6    labeling_0.3     stringi_1.1.5
[22] scales_0.4.1     backports_1.1.0  jsonlite_1.5
```