The World Atlas of Language Structures: Visualizing Set Intersections for Constituent Order Parameters

On the Limitations of Venn/Euler Diagrams and the Upshot of Up Sets

Maik Thalmann*

Göttingen; 08 May, 2020

Contents

1 Set Up R Environment

Set up some options and load the required packages for the current project. Chiefly among them, since they're the driving forces of the visualizations to come, are *eulerr* (Larsson 2020), *venneuler* (Wilkinson 2011), and *UpSetR* (Gehlenborg 2019).

Additionally, I will set a seed for random number generation. I am not quite positive about this, but suspect that the *eulerr* package used some random factors to compute the set alignments, as I have gotten quite varied results without an explicit seed.

```r
options(scipen = 999, width = 130)

packages <- c(
    # markdown
    "knitr", "kableExtra",
    # general
    "tidyverse", "dlookr", "janitor",
    # world map
```

* Georg-August-University Göttingen, maik.thalmann@gmail.com

```
    "rnaturalearth", "rnaturalearthdata", "sf", "hrbrthemes",
    # plotting
    "grid", "UpSetR", "eulerr", "venneuler"
)
xfun::pkg_attach(packages, install = TRUE)
set.seed(1234)
```

2 Data Preparation

2.1 Data Import

The data I will work with in the course of this project is from Dryer & Haspelmath (2013) (World Atlas of Language Structures, `https://wals.info`), and I will download the underlying data sets directly from the Github page where they're hosted (`https://github.com/cldf-datasets/wals`). To reduce computation times, I will subset it right from the very beginning to only focus on same parameters to do with constituent ordering. Note: while we want to check intersections within word order, we will disregard those languages with dual word order patterns (code 81B) to simplifiy result interpretation.

```
d <- "https://raw.githubusercontent.com/cldf-datasets/wals/master/cldf/values.csv"
d <- read.csv(d) %>%
    clean_names() %>%
    filter(parameter_id %in% c("81A", "87A", "88A", "89A")) %>%
    select(language_id, parameter_id, value) %>%
    droplevels()
```

Let's look at what we have so far. The excerpt can be found in Table 1.

```
head(d, 10) %>%
    kable(
        booktabs = T,
        caption = "Raw input data with only the parameters of interest."
    ) %>%
    kable_styling(latex_options = "HOLD_position")
```

Table 1: Raw input data with only the parameters of interest.

language_id	parameter_id	value
aab	81A	2
aab	87A	2
aab	88A	2
aab	89A	2
aar	87A	2
aar	88A	2
aar	89A	2
aba	81A	1
aba	88A	2
aba	89A	2

2.2 Data Check

Before proceeding, we need to make sure that there's no missing data. Additionally, we need to know what kinds of columms we're working with. Table 2 shows that we have four different parameters, at least one of which has 7 different possible values.

```
diagnose(d) %>%
    kable(
        booktabs = T,
        caption = "Data overview."
    ) %>%
    kable_styling(latex_options = "HOLD_position")
```

Table 2: Data overview.

variables	types	missing_count	missing_percent	unique_count	unique_rate
language_id	factor	0	0	1590	0.3104256
parameter_id	factor	0	0	4	0.0007809
value	integer	0	0	7	0.0013667

2.3 Transforming the Data

2.3.1 Pivotting

Because of the way that the different plotting methods we'll use later handle data, we need to transform it into wide format. In Table 3, you can see the output of this transformation: each parameter is now instantiated in its own column.

```
d <- d %>%
    pivot_wider(
        id_cols = "language_id",
        names_from = "parameter_id",
        values_from = "value"
    )
head(d, 10) %>%
    kable(
        booktabs = T,
        caption = "Wide-format data, where each column represents a language feature."
    ) %>%
    kable_styling(latex_options = "HOLD_position")
```

Table 3: Wide-format data, where each column represents a language feature.

language_id	81A	87A	88A	89A
aab	2	2	2	2
aar	NA	2	2	2
aba	1	NA	2	2
abi	2	3	1	NA
abk	1	2	1	3
abn	1	2	NA	NA
abo	1	2	NA	2
abu	2	2	2	2
abv	1	2	1	2
ace	7	2	2	1

2.3.2 Renaming

The values might be nicer if they were human-readable, so we'll add them in the next steps. The descriptions are contained in `codes.csv`. The relevant part is displayed in Table 4.

```
codes <- "https://raw.githubusercontent.com/cldf-datasets/wals/master/cldf/codes.csv"
codes <- read.csv(codes) %>%
    clean_names() %>%
    filter(parameter_id %in% c("81A", "87A", "88A", "89A")) %>%
    select(id, name) %>%
    pivot_wider(names_from = "id", values_from = "name")

codes %>%
    kable(
        booktabs = T,
        caption = "Parameter code descriptions."
    ) %>%
    kable_styling(latex_options = c("scale_down", "HOLD_position"))
```

Table 4: Parameter code descriptions.

81A-1	81A-2	81A-3	81A-4	81A-5	81A-6	81A-7	87A-1	87A-2	87A-3	87A-4	88A-1	88A-2	88A-3	88A-4	88A-5	88A-6	89A-1	89A-2	89A-3	89A-4
SOV	SVO	VSO	VOS	OVS	OSV	No dominant order	Adjective-Noun	Noun-Adjective	No dominant order	Only internally-headed relative clauses	Demonstrative-Noun	Noun-Demonstrative	Demonstrative prefix	Demonstrative suffix	Demonstrative before and after Noun	Mixed	Numeral-Noun	Noun-Numeral	No dominant order	Numeral only modifies verb

Below is the code to rename the parameters and add the descriptions. As the descriptions are very long at times, I opted for manual entry as opposed to simple replacements with the values from Table 4. For the outcome, see Table 5.

```
d <- d %>%
    rename(
        word_order = "81A", adj_noun = "87A", dem_noun = "88A", num_noun = "89A"
    ) %>%
    mutate(
        word_order = as.character(word_order),
        word_order = recode(word_order,
            "1" = "SOV", "2" = "SVO", "3" = "VSO",
            "4" = "VOS", "5" = "OVS", "6" = "OSV", "7" = "woNA"
        ),
        word_order = replace_na(word_order, "woNA"),
        adj_noun = as.character(adj_noun),
        adj_noun = recode(adj_noun,
            "1" = "ADJN", "2" = "NADJ", "3" = "adjNA"
        ),
        adj_noun = replace_na(adj_noun, "adjNA"),
        dem_noun = as.character(dem_noun),
        dem_noun = recode(dem_noun,
            "1" = "DemN", "2" = "NDem", "3" = "DemSx",
            "4" = "DemPx", "5" = "DemNDem", "6" = "demmixed"
        ),
        dem_noun = replace_na(dem_noun, "demNA"),
        num_noun = as.character(num_noun),
        num_noun = recode(num_noun,
            "1" = "NumN", "2" = "NNum", "3" = "numNA", "4" = "numNA"
        ),
        num_noun = replace_na(num_noun, "numNA"),
    )

head(d, 10) %>%
    kable(
        booktabs = T,
        caption = "Recoded input data where each column represents a super parameter relating to word order, and each column values is its parameter
    ) %>%
    kable_styling(latex_options = "HOLD_position")
```

Table 5: Recoded input data where each column represents a super parameter relating to word order, and each column values is its parameter setting.

language_id	word_order	adj_noun	dem_noun	num_noun
aab	SVO	NADJ	NDem	NNum
aar	woNA	NADJ	NDem	NNum
aba	SOV	adjNA	NDem	NNum
abi	SVO	adjNA	DemN	numNA
abk	SOV	NADJ	DemN	numNA
abn	SOV	NADJ	demNA	numNA
abo	SOV	NADJ	demNA	NNum
abu	SVO	NADJ	NDem	NNum
abv	SOV	NADJ	DemN	NNum
ace	woNA	NADJ	NDem	NumN

2.3.3 More Privots and Recoding

Here, we do some more pivotting and value replacements to get the data into the shape we need it to be. In particular, this means binary column values of either 1 or 0, depending on whether the parameter is expressed or not. This also means that, at the end of this process, all parameter settings (as opposed to the parameters themselves) need to be encoded as their own column, hence the pivotting. Unfortunately, there does not seem to be a way to apply `pivot_wider` to multiple columns at once without collapsing them, so we'll chain four pivots to achieve the desired outcome. We are left with the data in Table 6.

```
d <- d %>%
    pivot_wider(
        names_from = "word_order",
        values_from = "word_order",
        values_fill = list(word_order = 0)
    ) %>%
    pivot_wider(
        names_from = "adj_noun",
        values_from = "adj_noun",
        values_fill = list(adj_noun = 0)
    ) %>%
    pivot_wider(
        names_from = "dem_noun",
        values_from = "dem_noun",
        values_fill = list(dem_noun = 0)
    ) %>%
    pivot_wider(
        names_from = "num_noun",
        values_from = "num_noun",
        values_fill = list(num_noun = 0)
    )

d <- d %>%
    mutate_at(vars(!ends_with("_id")), function(x) as.numeric(x != "0"))

head(d, 10) %>%
    kable(
        booktabs = T,
        caption = "Widened Data containing binary identifiers regarding feature expression."
    ) %>%
    kable_styling(latex_options = c("scale_down", "HOLD_position"))
```

Table 6: Widened Data containing binary identifiers regarding feature expression.

language_id	SVO	woNA	SOV	VSO	VOS	OVS	OSV	NADJ	adjNA	ADJN	4	NDem	DemN	demNA	DemSx	DemPx	demmixed	DemNDem	NNum	numNA	NumN
aab	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0
aar	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0
aba	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
abi	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0
abk	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0
abn	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0
abo	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0
abu	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0
abv	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0
ace	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1

2.3.4 Add Language Data

Before finally turning our attention towards the actual visualizations, we will perform one (largely optional) step: adding some information about the 1590 languages whose parameters we're visualizing. This information won't actually be displayed in the set intersetion plots, but I think it makes the final data set more complete, so I'll add it anyway. Plus, it allows us to visualize our language sample on a world map, as you'll see in the next section.

As before, the language information is contained in yet another WALS data set, `languages.csv`. The columns of interest, namely those we will be adding to our reshaped data, are shown in Table 7.

```
langs <- "https://raw.githubusercontent.com/cldf-datasets/wals/master/cldf/languages.csv"
langs <- read.csv(langs) %>%
    clean_names() %>%
    select(id, name, latitude, longitude, family, genus) %>%
    rename(language_id = id)

head(langs, 10) %>%
    kable(
        booktabs = T,
        caption = "Available information about the languages in our data."
    ) %>%
    kable_styling(latex_options = "HOLD_position")
```

Table 7: Available information about the languages in our data.

language_id	name	latitude	longitude	family	genus
aab	Arapesh (Abu)	-3.450000	142.950000	Torricelli	Kombio-Arapesh
aar	Aari	6.000000	36.583333	Afro-Asiatic	South Omotic
aba	Abau	-4.000000	141.250000	Sepik	Upper Sepik
abb	Arabic (Chadian)	13.833333	20.833333	Afro-Asiatic	Semitic
abd	Abidji	5.666667	-4.583333	Niger-Congo	Kwa
abe	Arabic (Beirut)	33.916667	35.500000	Afro-Asiatic	Semitic
abh	Arabic (Bahrain)	26.000000	50.500000	Afro-Asiatic	Semitic
abi	Abipón	-29.000000	-61.000000	Guaicuruan	South Guaicuruan
abk	Abkhaz	43.083333	41.000000	Northwest Caucasian	Northwest Caucasian
abm	Alabama	32.333333	-87.416667	Muskogean	Muskogean

After combining the language information with our data, we are left with Table 8.

```r
d <- d %>%
    left_join(langs)

head(d, 10) %>%
    kable(
        booktabs = T,
        caption = "Final data including all necessary language information."
    ) %>%
    kable_styling(latex_options = c("scale_down", "HOLD_position"))
```

Table 8: Final data including all necessary language information.

language_id	SVO	woNA	SOV	VSO	VOS	OVS	OSV	NADJ	adjNA	ADJN	4	NDem	DemN	demNA	DemSx	DemPx	demmixed	DemNDem	NNum	numNA	NumN	name	latitude	longitude	family	genus
aab	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	Arapesh (Abu)	-3.45000	142.95000	Torricelli	Kombio-Arapesh
aar	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	Aari	6.00000	36.58333	Afro-Asiatic	South Omotic
aba	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	Abau	-4.00000	141.25000	Sepik	Upper Sepik
abi	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	Abipón	-29.00000	-61.00000	Guaicuruan	South Guaicuruan
abk	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	Abkhaz	43.08333	41.00000	Northwest Caucasian	Northwest Caucasian
abn	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	Arabana	-28.25000	136.25000	Pama-Nyungan	Central Pama-Nyungan
abo	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	Arbore	5.00000	36.75000	Afro-Asiatic	Lowland East Cushitic
abu	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	Abun	-0.50000	132.50000	West Papuan	North-Central Bird's Head
abv	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	Abui	-8.25000	124.66667	Timor-Alor-Pantar	Greater Alor
ace	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	Acehnese	5.50000	95.50000	Austronesian	Malayo-Sumbawan

3 Plots

3.1 World Map: sf

Using the *sf* package (Pebesma 2018), let's look at what kinds of languages we have data on (and to actually make use of the coordinate information). The output is shown in Figure 1.

```r
world <- ne_countries(
    scale = "small", returnclass = "sf",
    continent = c(
        "africa", "oceania", "asia",
        "europe", "north america", "south america"
    )
)

ggplot(data = world) +
    geom_sf(size = .1, fill = "gray95") +
    geom_point(
        data = d,
        aes(x = longitude, y = latitude),
        size = .5, alpha = .2, color = "#31497E"
    ) +
    labs(x = NULL, y = NULL)
```

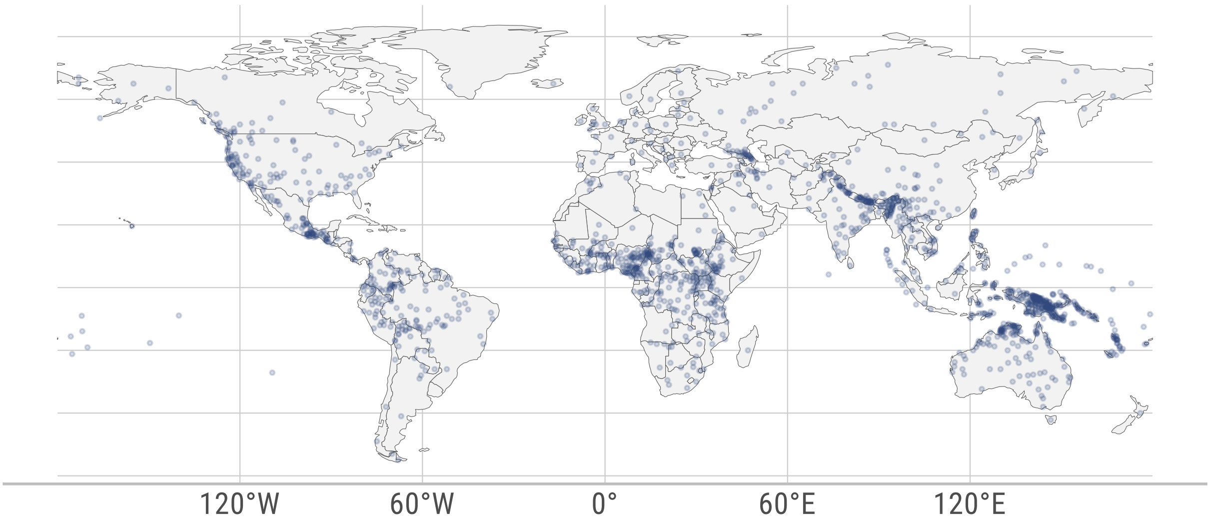

Figure 1: Languages considered in our dataset, created with the *sf* package.

3.2 Venn-Diagram 1: venneuler

Let's start with the first Venn/Euler diagram[1] (using the *venneuler* package, Wilkinson 2011). Note that in Figure 2 there does not seem to be a way of using ellipses instead of circles for the shape of the sets, which leads to awkward layout design and loss of intersection information in the present case.

```
sets <- d %>%
    select(SOV, SVO, OVS, OSV, NADJ, ADJN)

venn <- venneuler(as.data.frame(sets))
par(cex = .35)
plot(venn)
```

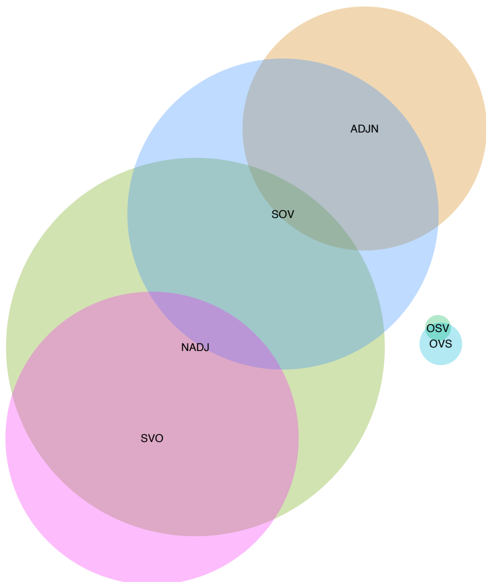

Figure 2: Venn/Euler diagram using *venneuler* package.

3.3 Venn-Diagram 2: eulerr

The second Venn/Euler diagram, Figure 3, was created with the *eulerr* package (Larsson 2020). Here, it is possible to use an ellipse as the basic set shape, which allows a more information rich display (note the OVS and OSV overlaps that could not be shown previously).

1 See https://en.wikipedia.org/wiki/Euler_diagram for a description of Euler diagrams, as well as differentiation from Venn diagrams.

You may have noticed that I opted for generating a legend as opposed to adding the set names directly within the ellipses. This way done primarily so that as many intersections as possible can be displayed. Of course, this comes at the price of readability. Different from Figure 2, the current plot cannot be interpreted as quickly, simply because a legend lookup is required (and, potentially, because my color choice may not be the best one out there).

```
plot(euler(
    as.data.frame(sets),
    shape = "ellipse"
),
quantities = list(
    type = "counts", fontsize = 8, font = 3
),
edges = list(col = "white", lex = .5, lty = 3),
fills = scico::scico(6, palette = "acton", begin = .2),
legend = list(TRUE, fontsize = 11)
)
```

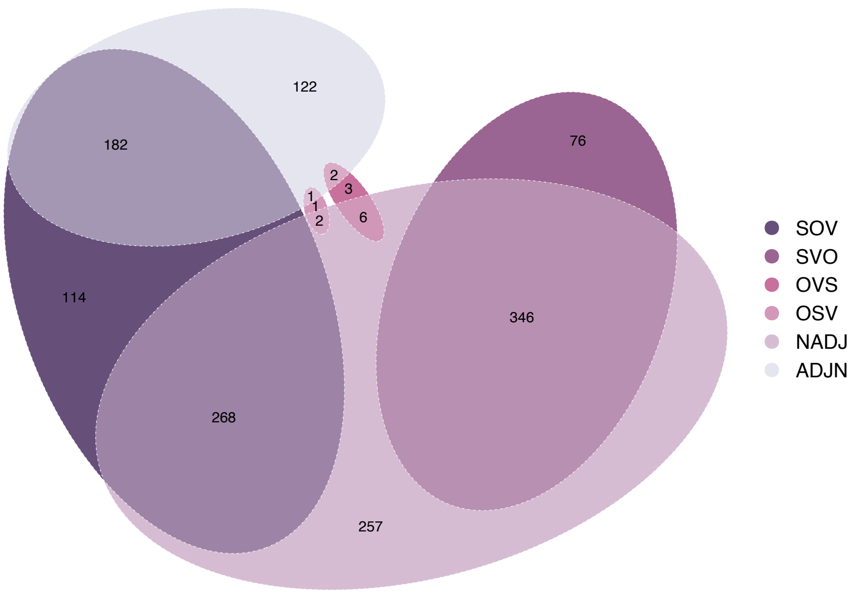

Figure 3: Venn/Euler diagram using the *eulerr* package.

The alternative, with the set names noted within the sets themselves, is shown in Figure 4. As before, while the larger sets are unaffected by this choice, the results for the smaller ones, OVS and OSV, are quite different.

As I hope to have demonstrated in detail, Venn/Euler diagrams, especially when considering a larger number of sets with varying sizes, may not be the best choice for set intersection visualizations, despite most people's familiarity with them.

```
plot(euler(
    as.data.frame(sets),
    shape = "ellipse"
),
quantities = list(
    type = "counts", fontsize = 8, font = 3
),
edges = list(col = "white", lex = .5, lty = 3),
fills = scico::scico(6, palette = "acton", begin = .2)
)
```

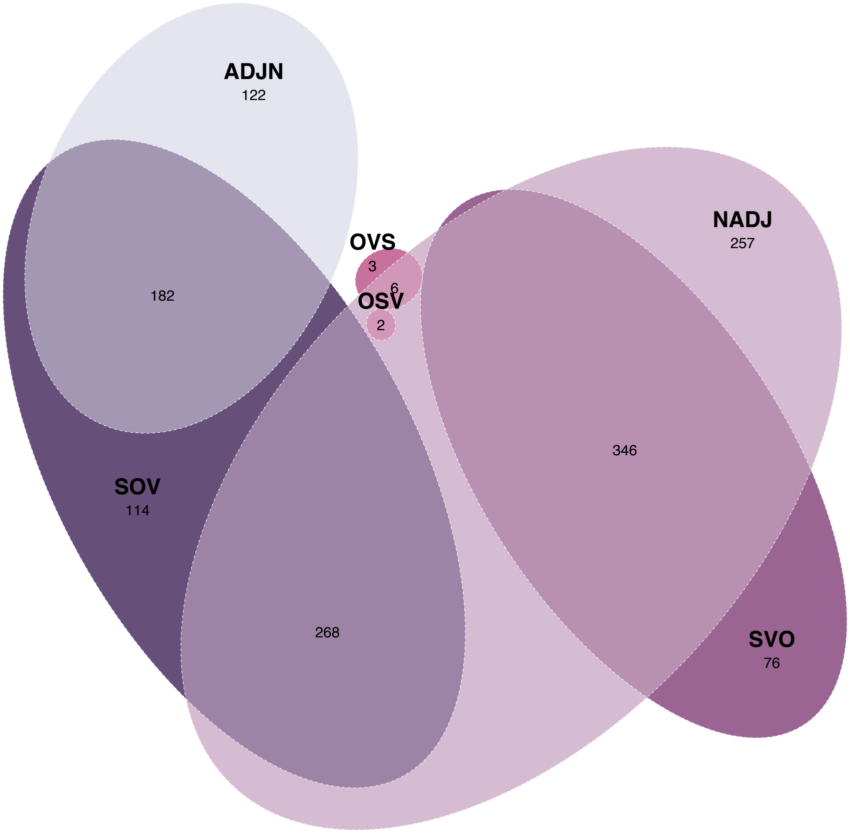

Figure 4: Venn/Euler diagram using the *eulerr* package.

3.4 Up Set Plot: UpSetR

Finally, to overcome some of the problems we encountered with Venn/Euler diagrams, we have the up set visualization in Figure 5, created with the *UpSetR* package (Gehlenborg 2019, see also Lex & Gehlenborg 2014). Here, because the basic type of plot is a bar diagram, all intersections can be displayed without needing to be concerned about the geometric shape of the sets as either ellipses or circles. This also allows for the inclusion of two other parameters: ordering of demonstratives with respect to the noun as well as numeral positioning.

```
vars <- c(
    "SOV", "SVO", "VOS", "VSO", "OVS", "OSV",
    "NADJ", "ADJN", "NDem", "DemN", "NNum", "NumN"
)

upset(
    as.data.frame(d),
    sets = vars,
    order.by = c("freq"),
    keep.order = TRUE,
    mainbar.y.label = "Word Order and Adjective Placement Dependencies",
    sets.x.label = "Number of Languages",
    mb.ratio = c(0.6, 0.4),
    queries = list(
        list(
            query = intersects,
            params = list("SOV", "NADJ"),
            color = "#1F7A80FF",
            active = T
        )
    ),
    text.scale = .75,
    shade.color = "#B2D2DEFF",
    main.bar.color = "#64A6BDFF",
    matrix.color = "#B1B8CA",
    sets.bar.color = "#F4CAE0FF"
)
```

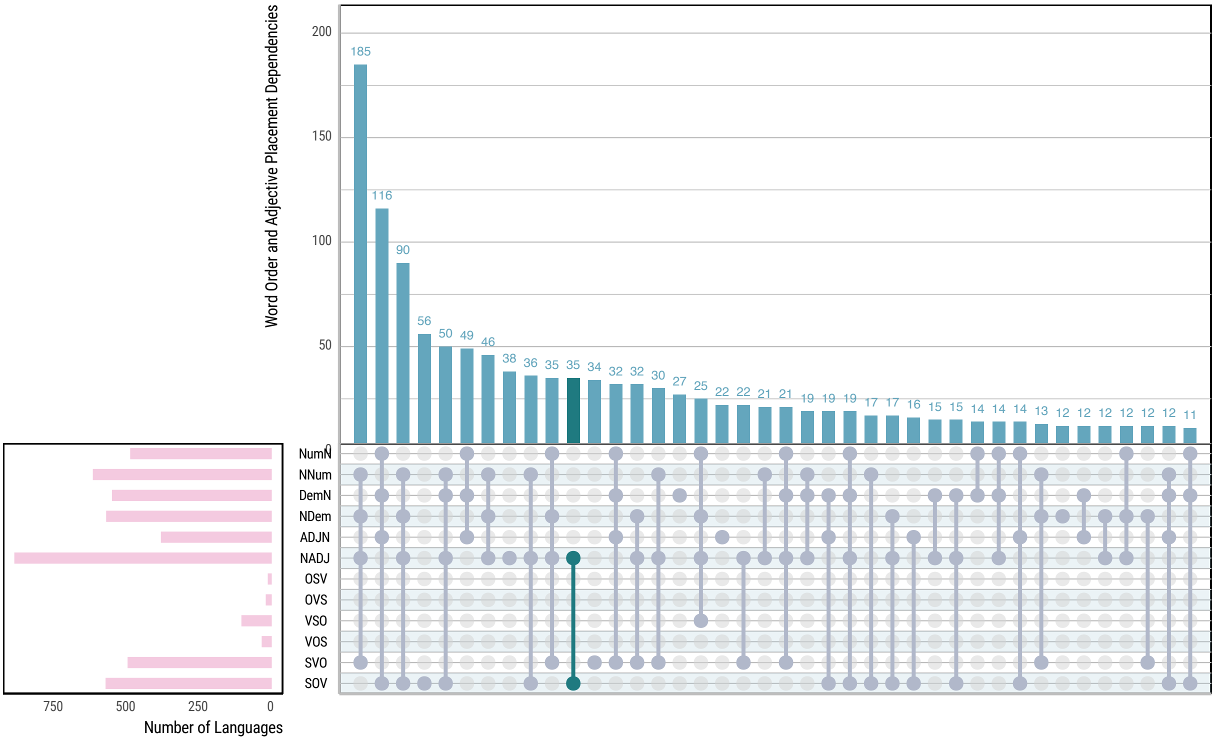

Figure 5: UpSet diagram representing the dependencies between constituent orders; created using the *UpSetR* package. The differently colored bar is simply there for code illustration purposes and can be ignored.

For comparison, Figure 6 is the up set version of the Venn/Euler diagrams (i.e., exculding demonstrative and numeral position).

```
upset(
    as.data.frame(d),
    sets = vars[1:8],
    order.by = c("freq"),
    keep.order = TRUE,
    mainbar.y.label = "Word Order and Adjective Placement Dependencies",
    sets.x.label = "Number of Languages",
    text.scale = .75,
    shade.color = "#B2D2DEFF",
    main.bar.color = "#64A6BDFF",
    matrix.color = "#B1B8CA",
    sets.bar.color = "#F4CAE0FF"
)
```

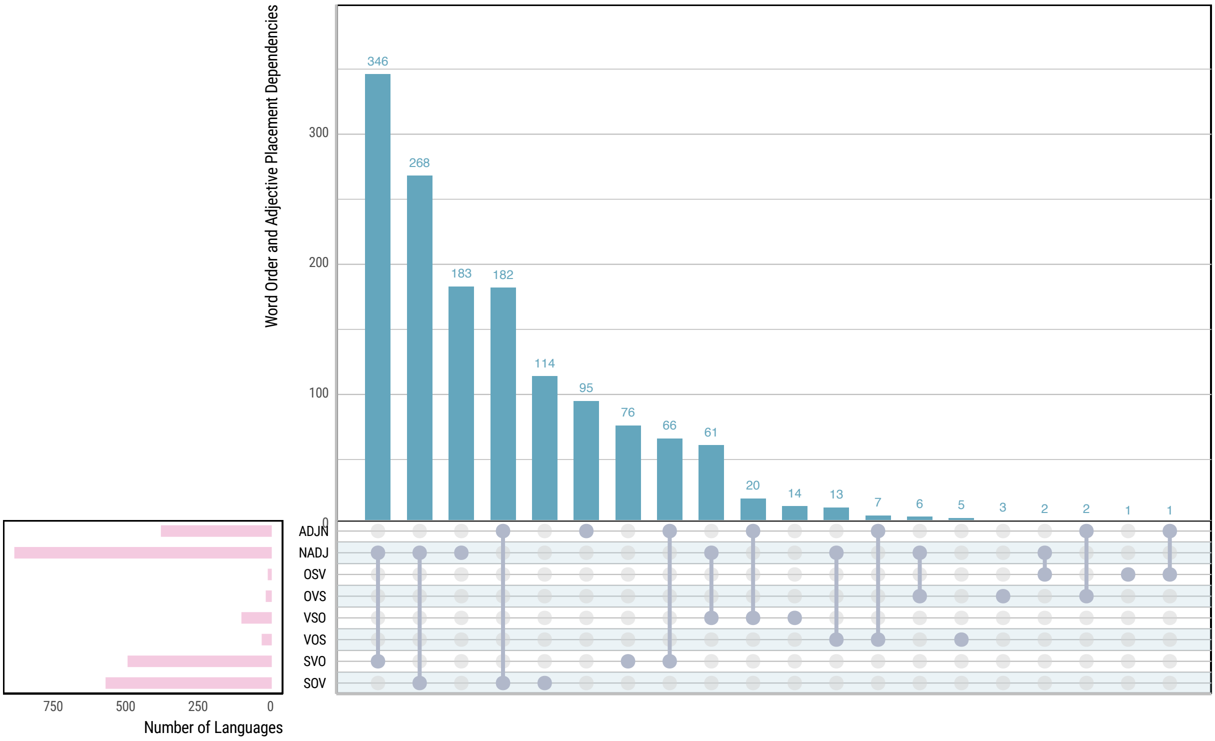

Figure 6: Up Set diagram using the same sets as the Venn/Euler diagrams.

References

Dryer, Matthew S. & Martin Haspelmath (eds.). 2013. *WALS online*. Leipzig: Max Planck Institute for Evolutionary Anthropology.

Gehlenborg, Nils. 2019. *UpSetR: A More Scalable Alternative to Venn and Euler Diagrams for Visualizing Intersecting Sets*. R package version 1.4.0.

Larsson, Johan. 2020. *eulerr: Area-proportional Euler and Venn Diagrams with Ellipses*. R package version 6.1.0.

Lex, Alexander & Nils Gehlenborg. 2014. Sets and intersections. *Nature Methods* 11(8). 779–779.

Pebesma, Edzer. 2018. Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal* 10(1). 439–446.

Wilkinson, Lee. 2011. *venneuler: Venn and Euler Diagrams*. R package version 1.1-0.

A Session Info

```
xfun::session_info(dependencies = FALSE)
R version 3.6.3 (2020-02-29)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS Catalina 10.15.4

Locale: en_US.UTF-8 / en_US.UTF-8 / en_US.UTF-8 / C / en_US.UTF-8 / en_US.UTF-8

Package version:
  venneuler_1.1-0       rJava_0.9-12             eulerr_6.1.0          UpSetR_1.4.0          hrbrthemes_0.8.0
  sf_0.9-2              rnaturalearthdata_0.1.0  rnaturalearth_0.1.0   janitor_2.0.1         dlookr_0.3.13
  mice_3.8.0            forcats_0.5.0            stringr_1.4.0         dplyr_0.8.5           purrr_0.3.4
  readr_1.3.1           tidyr_1.0.2              tibble_3.0.1          ggplot2_3.3.0         tidyverse_1.3.0
  kableExtra_1.1.0      knitr_1.28               readxl_1.3.1          backports_1.1.6       Hmisc_4.4-0
  corrplot_0.84         systemfonts_0.2.0        plyr_1.8.6            polylabelr_0.2.0      sp_1.4-1
  splines_3.6.3         digest_0.6.25            htmltools_0.4.0       gdata_2.18.0          fansi_0.4.1
  magrittr_1.5          checkmate_2.0.0          memoise_1.1.0         cluster_2.1.0         ROCR_1.0-7
  openxlsx_4.1.4        extrafont_0.17           modelr_0.1.6          R.utils_2.9.2         extrafontdb_1.0
  xts_0.12-0            sandwich_2.5-1           jpeg_0.1-8.1          colorspace_1.4-1      blob_1.2.1
  rvest_0.3.5           haven_2.2.0              xfun_0.13             tcltk_3.6.3           crayon_1.3.4
  jsonlite_1.6.1        libcoin_1.0-5            survival_3.1-12       zoo_1.8-7             glue_1.4.0
  polyclip_1.10-0       smbinning_0.9            gtable_0.3.0          webshot_0.5.2         scico_1.1.0
```

```
R.cache_0.14.0        car_3.0-7           Rttf2pt1_1.3.8      quantmod_0.4.17     abind_1.4-5
scales_1.1.0          mvtnorm_1.1-0       DBI_1.1.0           Rcpp_1.0.4.6        viridisLite_0.3.0
xtable_1.8-4          htmlTable_1.13.3    units_0.6-6         foreign_0.8-76      bit_1.1-15.2
Formula_1.2-3         sqldf_0.4-11        htmlwidgets_1.5.1   httr_1.4.1          gplots_3.0.3
RColorBrewer_1.1-2    acepack_1.4.1       ellipsis_0.3.0      farver_2.0.3        pkgconfig_2.0.3
R.methodsS3_1.8.0     nnet_7.3-13         dbplyr_1.4.3        labeling_0.3        tidyselect_1.0.0
rlang_0.4.5           munsell_0.5.0       cellranger_1.1.0    tools_3.6.3         cli_2.0.2
gsubfn_0.7            generics_0.0.2      moments_0.14        RSQLite_2.2.0       broom_0.5.6
evaluate_0.14         yaml_2.2.1          bit64_0.9-7         fs_1.4.1            zip_2.0.4
caTools_1.18.0        nlme_3.1-147        R.oo_1.23.0         xml2_1.3.1          compiler_3.6.3
rstudioapi_0.11       curl_4.3            png_0.1-7           e1071_1.7-3         reprex_0.3.0
stringi_1.4.6         rgeos_0.5-2         gdtools_0.2.2       lattice_0.20-41     Matrix_1.2-18
classInt_0.4-3        styler_1.3.2        vctrs_0.2.4         RcmdrMisc_2.7-0     pillar_1.4.3
lifecycle_0.2.0       data.table_1.12.8   bitops_1.0-6        R6_2.4.1            latticeExtra_0.6-29
KernSmooth_2.23-16    gridExtra_2.3       rio_0.5.16          MASS_7.3-51.5       gtools_3.8.2
assertthat_0.2.1      chron_2.3-55        proto_1.0.0         withr_2.2.0         nortest_1.0-4
DMwR_0.4.1            hms_0.5.3           prettydoc_0.3.1     rpart_4.1-15        class_7.3-16
rmarkdown_2.1         inum_1.0-1          snakecase_0.11.0    carData_3.0-3       TTR_0.23-6
partykit_1.2-7        lubridate_1.7.8     base64enc_0.1-3     tinytex_0.22
```