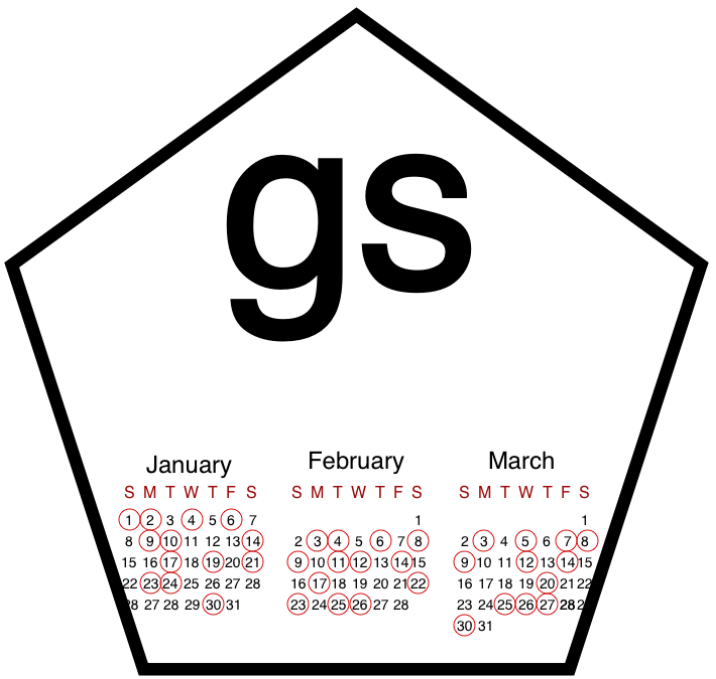
Introducing `{gs}`

A grammar of recurring calendar events

James Laird-Smith

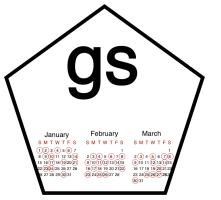
EARL Conference

12 September 2019



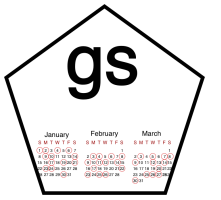
EARL
ENTERPRISE APPLICATIONS OF THE R LANGUAGE

Why `{gs}`?



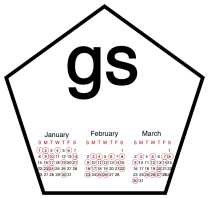
- R already has tools for dates and times.
  - Eg. `base`, `lubridate` etc.
  - These can create evenly spaced events.
- R has few tools for schedules **when the event pattern is irregular**.
  - Eg. Summer Bank Holiday (last Monday in August)
  - Easter
  - Strange bespoke examples.
- Often important (like financial contracts).

What is {gs}?



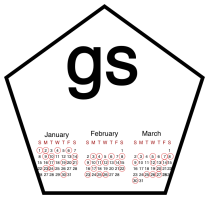
- **An R package** for recurring calendar events (schedules).
- Functions for:
  - Creating schedules.
  - Composing schedules.
  - Working with schedules.
  - (The whole schedule lifecycle.)
- **A grammar**
  - Solve your own problems.

What can {gs} do?



- Simple schedules
- Holiday shortcuts (i.e. Easter)
- Nth (last) occurrence in a period (eg. Bank holidays)
- Directional schedules
  - Eg. after US Thanksgiving but before Black Friday
- Incremental schedules
  - Eg. every 5 business days
- Forthcoming
  - More holidays (in their own package)
  - Rolling dates

A introductory example




```
library(gs)
library(magrittr)
```

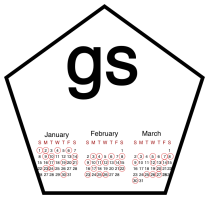
- How would you schedule New Year's Day?
- What is special about New Year's Day?
    - The first day of the year.

`on_yday()` is a dedicated function to make a schedule of events occurring on certain days of the year.

```
on_yday(1)
```

```
## A schedule of events containing 1 term(s)
```

Basic schedules



- Built on `lubridate` accessor functions:

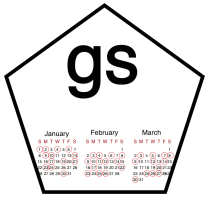
```
lubridate::yday(as.Date("2000-01-01"))
```

```
## [1] 1
```

```
on_yday(1)
```

```
## A schedule of events containing 1 term(s)
```

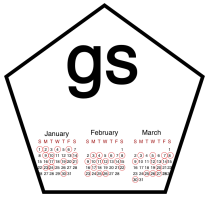
Basic schedules (cont.)



- **`lubridate`** has date accessor functions.
- Most have equivalent **`gs`** schedules.

lubridate::	**gs::**
wday()	on_wday()
mday()	on_mday()
qday()	on_qday()
yday()	on_yday()
week()	in_week()
month()	in_month()
quarter()	in_quarter()
semester()	in_semester()
year()	in_year()

Using schedules



- Schedules can be made into objects:

```
on_new_years_day <- on_yday(1)
```

- Once you have a schedule object, you can test whether certain dates fall on it. This is done using the **`happen()`** function.

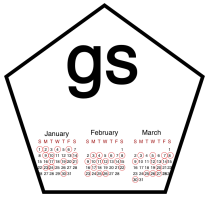
```
my_dates <- seq.Date(from = as.Date("2000-01-01"),
                     to = as.Date("2000-01-05"),
                     by = "1 day")

my_dates
```

```
## [1] "2000-01-01" "2000-01-02" "2000-01-03" "2000-01-04" "2000-01-05"
```

```
happen(on_new_years_day, my_dates)
```

```
## [1]  TRUE FALSE FALSE FALSE FALSE
```

Using schedules (cont.)



- Once we have a schedule object, we can get the events from it using the **`schedule_days()`** function.

```
schedule_days(on_new_years_day)
## Produces an error!
```

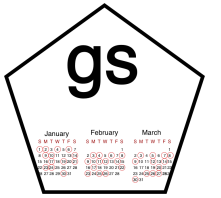
```
schedule_days(on_new_years_day,
              from = as.Date("2000-01-01"),
              to = as.Date("2004-12-31"))
```

```
## [1] "2000-01-01" "2001-01-01" "2002-01-01" "2003-01-01" "2004-01-01"
```

```
schedule_days(on_new_years_day, from = 2000, to = 2004)
```

```
## [1] "2000-01-01" "2001-01-01" "2002-01-01" "2003-01-01" "2004-01-01"
```

Composing schedules



- Say you now wanted to create a schedule for Christmas Day.
- What is special about Christmas Day?
  - Use **`on_yday()`** again? Is it the 359th day of the year? Not always. Not in a leap year where it is the 360th.
  - It is always the 25th of December.

```
in_december <- in_month("Dec")  # All days in December!
on_twenty_fifth <- on_mday(25)  # The 25th day of every month!
```

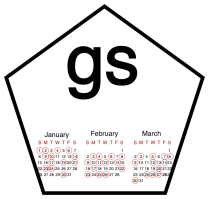
- There is a function to find the intersection of two schedules called **`only_occur()`**.

```
on_christmas_day <- only_occur(in_month("Dec"), on_mday(25))
```

```
schedule_days(on_christmas_day, from = 2000, to = 2004)
```

```
## [1] "2000-12-25" "2001-12-25" "2002-12-25" "2003-12-25" "2004-12-25"
```

Composing schedules (2)



- Say I wanted only Christmas Days occurring on Sunday:

```
on_christmas_day <- only_occur(in_month("Dec"), on_mday(25))
```

```
on_christmas_sunday <- only_occur(on_christmas_day, on_wday("Sun"))
```

- Getting the occurrences this millennium.

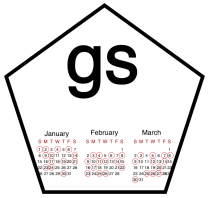
```
schedule_days(on_christmas_sunday, from = 2000, to = 2019)
```

```
## [1] "2005-12-25" "2011-12-25" "2016-12-25"
```

I could also use the pipe (**%>%**) operator to get the same result:

```
on_christmas_sunday <-
  on_mday(25) %>%
  only_occur(in_month("Dec")) %>%
  only_occur(on_wday("Sun"))
```

Composing schedules (3)



- What if we wanted to do the reverse, which is to combine the events of two schedules.
- For this there is the **`also_occur()`**

```
on_public_holidays <- also_occur(on_christmas_day, on_new_years_day)
```

You can do the same as before:

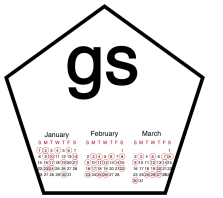
```
schedule_days(on_public_holidays, from = 2019, to = 2020)
```

```
## [1] "2019-01-01" "2019-12-25" "2020-01-01" "2020-12-25"
```

Thank you!

Questions?

Appendix A



What makes `gs` a grammar?

> A grammar is a framework that lays out the minimal set of independent components and a means of composing them to solve a wide range of problems within a domain.
>
> - *Hadley Wickham*[1]

{gs} is a minimal set of independent **schedules** and a means of composing them to make a wide range of **patterns for recurring calendar events.**

[1] Wickham, H., 2015. Teaching Safe-Stats, Not Statistical Abstinence. online supplement discussion of “Mere Renovation is Too Little Too Late: We Need to Rethink Our Undergraduate Curriculum from the Ground Up” by G. Cobb, *The American Statistician*, 69.