The Journal of Open Source Software

set6: R6 Mathematical Sets Interface

Raphael Sonabend[1] **and Franz J. Kiraly**[1]

1 Department of Statistical Science, University College London

DOI: 10.21105/joss.02598

Software

- Review
- Repository
- Archive

Editor: David P. Sanders

Reviewers:

- @dmbates
- @davidjohannesmeyer
- @davidjohannesmeyer
- @wch

Submitted: 16 June 2020
Published: 08 November 2020

Summary

`set6` makes use of the R6 object-oriented paradigm to introduce classes for important mathematical objects, including sets, tuples, and intervals (finite and infinite). Until now, the `R` (R Core Team, 2017) programming language has traditionally supported mathematical sets in one of two ways: 1. via the five set operation functions: `union`, `intersect`, `setdiff`, `setequal`, `is.element`; and 2. via the `sets` (Meyer & Hornik, 2009) package. `set6` uses `R6` (Chang, 2018) and has a clear class interface with minimal dependencies, which makes it the perfect dependency for any package that requires mathematical data types as R6 objects. Making use of design patterns (Gamma et al., 1996), such as wrappers and compositors, `set6` allows for symbolic representation of sets to ensure maximum efficiency, and to provide neat and clear print methods.

`set6` is currently being used in `distr6` (Sonabend & Kiraly, 2019), which is an object-oriented probability distributions interface, that makes use of `set6` for distribution and parameter support. Additional uses of `set6` include representing infinite sets, and constructing assertions.

The speed and efficiency of `R6` and `Rcpp` (Eddelbuettel & Francois, 2011) allows `set6` to be a scalable and efficient interface. A focus on symbolic representation and neat printing methods means that `set6` can accurately and clearly represent complicated sets. `set6` has the ambitious long-term goal of being the only dependency package needed for object-oriented interfaces in `R` that require clear symbolic representations of mathematical sets.

Related software in `R` includes `sets` (Meyer & Hornik, 2009), which uses the S3 and S4 object-oriented paradigms to implement mathematical sets. Whilst `sets` and `set6` have similar implemented features, as both offer multiple forms of sets (intervals, fuzzy, etc.) and with symbolic representation, the `sets` package does not offer lazy evaluation in set operations, which leads to significant overheads for large or possibly-infinite sets. Similar packages in other computing languages include: i) `portion` (Decan, 2020) in Python, which only supports intervals without generalisation to other mathematical sets; ii) `IntervalSets.jl` (Navelkar, 2016) in Julia, which is again limited to intervals though with good features for infix operators and inspection; iii) a variety of packages in the `JuliaIntervals` suite (https://juliapackages.com/u/juliaintervals), which primarily focus on rigorous arithmetic implementation; and iv) `LazySets.jl` (Forets & Schilling, 2018) and `DomainSets.jl` (Huybrechs, 2018) which both faciliate symbolic representation of sets and infinite intervals in Julia.

The example below demonstrates construction of a set and interval, comparisons of these, the set complement operator, and printing of the final result. Note the example does not use unicode printing but that this is possible within the package.

```
> a <- Set$new(1, 2, 3)
> a$print()
{1, 2, 3}
> a$contains(c(1, "a"))
[1] TRUE FALSE
```

```
> b <- Interval$new(1, Inf, class = "numeric")
> b$isSubset(a)
TRUE
> c <- b - a
> c$print()
(1,2) U (2,3) U (3,+Inf]
```

Key Design Principles

1. **Maximum user-control over set operations** - Users can select how operations act on sets, including a choice of associativity, lazy evaluation, and unicode printing.
2. **Minimal dependencies** - `set6` has the goal of being a key dependency to any object-oriented package requiring representation of mathematical sets, for example for representing function inputs and supports. Therefore `set6` is itself dependent on only three packages.
3. **Inspectability and reactive user interface** - `set6` prioritises symbolic representation and lazy evaluation to allow for the package to be scalable and to fit into any other package. However it is ensured that this does not detract from a clear user interface, i.e. at all times it should be clear what an object contains both externally (how it prints) and internally (inspection methods). `set6` allows sets to be queried in many different ways, including calling the elements in the set (if finite), finding the bounds of the set (if numeric), and listing properties and traits.

```
# Symbolic representation allows neat representation of infinite sets
# and intervals whilst making the elements clear by inspection

> I <- Interval$new(10, Inf, type = "[)")
> I$print()
[10,Inf)
> I$contains(9:11)
[1] FALSE TRUE TRUE

> n <- Naturals$new()
> n$print()
NO
# binary operators also available
> c(pi, 2) %inset% n
[1] FALSE TRUE
```

4. **Combination of lazy and greedy evaluation** - By default, 'multiplying' operations such as products and powersets are evaluated lazily, whereas 'adding' operations such as unions and differences are evaluated greedily. These prevent system crashes from trying to evaluate sets of very large cardinality. In all cases, the user can override defaults. Symbolic representation is used with lazy evaluation so that large sets can be printed neatly without the individual elements ever being evaluated.

```
# Lazy evaluation allows very large sets to be queried without
# explicit evaluation

> s <- Set$new(1, 2, 3)
> p <- powerset(powerset(powerset(s)))
```

```
> p$print()
2^2^2^{1, 2, 3}
> p$properties$cardinality
[1] 1.157921e+77
> p$contains(Set$new(Set$new(Set$new(1), Set$new(1, 2, 3))))
[1] TRUE
```

Key Use-Cases

1. **Constructing and querying mathematical sets** - Many mathematical Set-like objects can be constructed, including sets, tuples, intervals, and fuzzy variants. Sets can contain objects of any `R` type that have a valid `as.character` coercion method that creates a unique symbolic representation of the class; this is required as internally checks are performed symbolically on character representations.
2. **Containedness checks** - Public methods allow all objects inheriting from `Set` to check if elements are contained within them. This provides a powerful mechanism for use with parameter or distribution supports for other packages as it can be viewed as a 'type check', i.e. checks if a value fits within a specified mathematical type. A C++ implementation of these checks in Rcpp (Eddelbuettel & Francois, 2011) means that the computations are incredibly quick for sets and intervals of any size.
3. **Representation of infinite sets** - Symbolic representation and lazy evaluation allows infinite (or very large) sets and intervals to be constructed. This also allows operations such as powerset to be used without crashing the system.
4. **Comparison of, possibly infinite, sets** - Two `Set` objects can be compared to check if they are equal or (proper) sub/supersets. Infix operators allow quick and neat comparison.
5. **Creation of composite sets from simpler classes** - Common set operations, such as unions and complements are implemented, as well as products and exponents. These make use of S3 dispatch to allow quick calculation of composite sets. In line with design principle 4), lazy and greedy evaluation with symbolic representation allow for composite sets to be created, inspected, and printed, without ever needing to be evaluated themselves.

Software Availability

`set6` is available on GitHub and CRAN. It can either be installed from GitHub using the `devtools` (Wickham et al., 2019) library or directly from CRAN with `install.packages`. The package uses the MIT open-source licence. Contributions, issues, feature requests, and general feedback can all be found and provided on the project GitHub. Full tutorials and further details are available on the project website.

References

Chang, W. (2018). *R6: Classes with Reference Semantics*. https://cran.r-project.org/package=R6

Decan, A. (2020). *Portion - data structure and operations for intervals*. PyPi. https://pypi.org/project/portion/

Eddelbuettel, D., & Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software*, *40*(8), 1–18. https://doi.org/10.18637/jss.v040.i08

Forets, M., & Schilling, C. (2018). *LazySets.jl*. https://github.com/JuliaReach/LazySets.jl

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1996). *Design Patterns: Elements of Reusable Software* (p. 395). https://doi.org/10.1093/carcin/bgs084

Huybrechs, D. (2018). *DomainSets.jl*. https://github.com/JuliaApproximation/DomainSets.jl

Meyer, D., & Hornik, K. (2009). Generalized and Customizable Sets in R. *Journal of Statistical Software*, *31*(2), 1——27. https://doi.org/10.18637/jss.v031.i02

Navelkar, A. (2016). *IntervalSets.jl*. https://github.com/JuliaMath/IntervalSets.jl

R Core Team. (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing.

Sonabend, R., & Kiraly, F. (2019). *distr6: The Complete R6 Probability Distributions Interface*. CRAN. https://cran.r-project.org/package=distr6

Wickham, H., Hester, J., & Chang, W. (2019). *devtools: Tools to Make Developing R Packages Easier*. CRAN.